Niko Koskimaa

Järjestelmävalvonnan kehittäminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

15.4.2018

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Niko Koskimaa Järjestelmävalvonnan kehittäminen 37 sivua 15.4.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Diplomi-insinööri Jussi Sainio Lehtori Simo Silander

Opinnäytetyö jakautuu kahteen osaan. Ensimmäisessä osassa Sensisto Oy:n käyttöön valittiin (verkon) valvontajärjestelmä ja kehitettiin sitä varten valvontaskriptejä. Toisessa osiossa kehitettiin yksinkertainen Arduino-prototyyppi humidorin kosteuden ja lämpötilan valvontaa varten.

Sensiston järjestelmien määrän kasvettua yritys tarvitsi toimintavakauden ja asiakastyytyväisyyden parantamiseksi käyttöönsä valvontajärjestelmän. Yrityksellä oli käytössään väliaikainen järjestelmä ennen työn aloittamista. Väliaikaisen järjestelmän tilalle asennettiin toinen järjestelmä ja valvonnan kattavuutta parannettiin. Yrityksen itse kehittämiä ohjelmia varten kehitettiin valvontaskriptejä, joilla niitä alettiin valvoa.

Opinnäytetyön loputtua yrityksen valvonta kattoi lähes kaikki järjestelmät. Joitain järjestelmiä ei pystytty lisäämään valvonnan piiriin. Joitain ohjelmia jäi valvonnan ulkopuolelle, koska ohjelmien tiloja ei voinut selvittää ohjelman ulkopuolelta. Valvontajärjestelmää pystyttiin hyödyntämään viestinnässä asiakkaiden kanssa ja sen avulla löydettiin myös muistivuoto eräästä yrityksen ohjelmasta.

Prototyyppiosiossa kehitettiin Arduino Unolla kosteutta ja lämpötilaa mittaava etäluettava mittari. Mittari kehitettiin, jotta voitiin testata suoraan tietokoneeseen kytkettävien laitteiden valvontaa valvontajärjestelmällä. Mittarilla oli tarkoitus valvoa humidorin kosteutta. Valvontaa varten kehitettiin muutama pieni ohjelma ja valvontaskripti, joilla valvonta toteutettiin. Tulevaisuudessa mittarin alusta vaihdetaan fyysiseltä koolta pienempään, jotta se mahtuu täyden humidorin sisään.

Molempien projektin osien tavoitteet saavutettiin ja lopputuloksiin oltiin tyytyväisiä.

Avainsanat	Arduino, Linux, Python, Valvonta

Author(s) Title Number of Pages Date	Niko Koskimaa Monitoring System Development 37 pages 15 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Jussi Sainio, Master of Science (Technology) Simo Silander, Senior Lecturer

This thesis consists of two parts. In the first part a network monitoring system was chosen and deployed for Sensisto Ltd. Some monitoring scripts were developed for the chosen system. The second part of the project was to develop an Arduino prototype for measuring the humidity and temperature of a humidor.

The number of computer systems Sensisto Ltd. had had increased and needed a monitoring system to increase operating stability and customer satisfaction. The company had a temporary monitoring system in-place before the start of this thesis. The temporary solution was replaced with a different monitoring system and monitoring coverage was improved. Monitoring scripts were developed to monitor applications developed by the company.

At the end of the project almost all systems were covered by the monitoring system. Some applications could not be monitored, because the state of the programs could not be determined from outside of the applications. The company was able to make use of the monitoring system in customer communications and a memory leak was discovered in one of the company's applications with the monitoring system.

In the prototype part of the project Arduino Uno was used to create a remotely readable humidity and temperature sensor. The sensor was developed to test how the monitoring system would work for monitoring devices directly connected to computers. The sensor was created to monitor a humidor's humidity. Few small programs and a monitoring script were developed for monitoring the humidity. In the future the platform will be change from Arduino Uno to physically smaller sized platform so that it can be put into a full humidor.

Both parts of the project were successful, and the results were satisfactory.

Keywords	Arduino, Linux, Monitoring, Python

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta.
IDE	Integrated Development Environment. Ohjelmistojen kehitysympäristö.
JSON	JavaScript Object Notation. Yksinkertainen tiedostomuoto tiedonvälitykseen.
OOM	Out-of-memory. Tila, jossa käyttöjärjestelmä ei voi enää allokoida muistia prosessille.
STDOUT	Standard output. Unix- ja Linux-käyttöjärjestelmien standardivirta tulosteille.
STDERR	Standard error. Unix- ja Linux-käyttöjärjestelmien standardivirta virhetulosteille.
SSH	Secure Shell. Salattu etäyhteysprotokolla.
ZMQ	Zero Message Queue. Viestijonokirjasto.

1 Johdanto

Insinöörityö tehtiin Sensisto Oy:lle. Sensisto tuottaa analytiikkapalveluita ostoskeskuksille ja kaupoille. Analytiikkaa varten kerätään tietoa erilaisista lähteistä. Osa tiedoista kerätään hyödyntäen erilaisia antureita. Sensisto integroi ja asentaa antureita osana analytiikkapalvelujen tuottamista.

Yrityksen tuottamien palveluiden ja sisäisten järjestelmien määrän kasvettua järjestelmävalvonnan kysymys nousi esille yrityksen sisällä. Yritystoiminnalle kriittisiä järjestelmiä ei pystytty enää valvomaan ihmisvoimin, ja viive virheen syntyhetken ja sen huomaamisen välillä tahdottiin saada mahdollisimman lyhyeksi. Ratkaisuksi ongelmaan ehdotettiin (verkon) valvontajärjestelmän (eng. network monitoring system) käyttöönottamista.

Valvontajärjestelmillä voidaan automaattisesti kerätä tietoa järjestelmien tiloista ja hälyttää havaituista ongelmista ylläpitoa. Valvonnalla haluttiin myös ennaltaehkäistä kriittisten virheiden tapahtumista yrityksen tuottamissa palveluissa ja järjestelmissä siltä osin kuin mahdollista. Järjestelmävalvonnan toivottiin myös ratkaisevan tapahtumien automaattinen kirjanpito, jotta tarvittaessa tiedot menneistä tapahtumista olisivat löydettävissä.

Tietojärjestelmät ovat nykyaikaisissa yrityksissä ja muissa organisaatioissa usein erittäin tärkeässä asemassa. Järjestelmien toimintahäiriöt voivat vaikuttaa suoraan yrityksen liikevaihtoon [1]. Tietojärjestelmien tärkeyden vuoksi monet yritykset käyttävät paljon rahaa toimintavakauden takaamiseen. Valvonnalla voidaan pyrkiä ennalta ehkäisemään ongelmien syntymistä. [2].

Yrityksen täysin sisäisten tarpeiden lisäksi valvonnalla tahdottiin yrittää parantaa asiakastyytyväisyyttä. Huomaamalla ongelmia palveluissa ennen asiakkaita ja proaktiivisesti korjaamalla löydetyt ongelmat asiakas ei parhaimmillaan huomaa palvelussa olleita ongelmia ollenkaan. Ajateltiin, että proaktiivisesti korjaamalla ongelmia säästetään käyttäjien ja palveluntarjoajan aikaa sekä hermoja. Asiakkaan ei tarvitsisi ottaa yhteyttä palveluntarjoajaan, koska ongelmat on jo korjattu, eikä palveluntarjoajan tarvitsisi käyttää aikaa ongelmailmoitusten selvittämiseen. Ongelmista, joita ei pystyttäisi korjaamaan pikaisesti, voitaisiin ilmoittaa asiakkaille. Näin asiakkaat olisivat tietoisia mahdollisista palveluissa olevista ongelmista ja pysyisivät toivottavasti tyytyväisempinä palveluun.

Tärkeimpiä järjestelmiä varten yritys oli ottanut käyttöön väliaikaisen valvontajärjestelmän jo ennen insinöörityön aloittamista. Väliaikaista järjestelmää ei ollut konfiguroitu kunnolla, joten järjestelmä tuotti liian paljon viestejä. Suuren viestimäärän takia viestejä ei läpikäyty kunnolla, eivätkä viestit johtaneet toimenpiteisiin. Tarvittuja ominaisuuksia ei ollut määritelty kunnolla, joten järjestelmän vaatimusten läpikäynti ja uudelleenmäärittely oli osa työtä.

Insinöörityön valvontajärjestelmän valintaosion tavoitteina oli valita horisontaalisti skaalautuva valvontajärjestelmä yrityksen käyttöön, kasvattaa valvonnan kattavuutta ja kehittää tarvittaessa valvontaskriptejä valittua järjestelmää varten. Jatkokehitystä varten oli myös tärkeää, että järjestelmän tuottamia viestejä voidaan helposti hyödyntää esimerkiksi asiakkaille lähetettävissä vikatiedotteissa. Valvontajärjestelmän hyödyntäminen asiakkaalle lähetettävien vikatiedotteiden laatimisessa on tarkoitus toteuttaa vasta projektin toisessa vaiheessa. Lisäksi metriikkatiedon keräämiselle valvontajärjestelmän avulla ei nähty vielä tarvetta. Yrityksen käytössä on vain Linux-pohjaisia käyttöjärjestelmiä, joten yhteensopivuudesta muiden käyttöjärjestelmien kanssa ei juuri välitetty.

Opinnäytetyöhön kuului lisäksi valvontajärjestelmään kytkettävissä oleva kosteus- ja lämpömittarin prototyypin suunnittelu ja toteuttaminen. Prototyypin avulla haluttiin testata, voidaanko valvontajärjestelmällä valvoa suoraan tietokoneeseen kytkettäviä antureita järkevästi. Testaaminen oltaisiin voitu tehdä myös valmiilla USB-kosteusanturilla, mutta valmiit ratkaisut olivat kalliita. Prototyyppi tehtiin humidorin kosteuden ja lämpötilan monitoroimista varten.

2 Valvontajärjestelmät

Tässä työssä valvontajärjestelmällä, kutsutaan arkikielessä myös monitorointijärjestelmiksi, tarkoitetaan niin sanottuja verkonvalvontajärjestelmiä, -työkaluja ja -alustoja. Järjestelmästä riippuen nimeämiskäytäntö vaihtelee eikä ole täysin vakiintunut edes englannin kielessä. Näitä järjestelmiä kutsutaan joskus myös IT-infrastruktuurin valvontajärjestelmiksi. Myös järjestelmien komponenttien nimeämiskäytännöt vaihtelevat järjestelmien välillä.

Valvontajärjestelmiä voidaan hyödyntää moneen tarkoitukseen, mutta valvontajärjestelmien tärkeimpänä tarkoituksena voidaan pitää verkkoon kytkettyjen laitteiden ja järjestelmien toiminnan valvontaa ja havaituista ongelmista hälyttämistä. [1.]

Laitteiden ja järjestelmien toimivuuden seurannan lisäksi suurimmalla osalla valvontajärjestelmistä voidaan kerätä talteen metriikkatietoja. Kerättyä tietoa voidaan hyödyntää mm. kapasiteetin riittävyyden, laitteiden käyttöiän ja vikaantumisen ennustamiseen. Valvontajärjestelmät voivat myös itsessään hyödyntää kerättyä tietoa ongelmien havaitsemisessa.

Valvontajärjestelmät koostuvat joukosta ohjelmistokomponentteja. Järjestelmissä on yleensä minimissään valvontapalvelin, -agentteja ja -skriptejä. Valvontajärjestelmiin voidaan liittää myös muita komponentteja. Muita komponentteja ovat mm. valvonnassa hyödynnettävät välityspalvelimet ja valvontajärjestelmiin kytkettävät web-käyttöliittymät.

Valvontapalvelimen tehtävinä on monitoroitavien laitteiden kirjanpito, tarkastuksien ajamisen ajastaminen, tarkastuksien tulosten käsittely ja ylläpidon hälyttäminen. Laitteiden kirjanpidossa pidetään yllä tietoa järjestelmiin liitetyistä laitteista, laitteiden tiedoista ja niihin liittyvistä tarkastuksista. Osassa valvontajärjestelmiä valvontapalvelimia voi olla useampia klusterissa tai vikasietoisuuden varalta.

Valvontajärjestelmien toiminta pohjautuu tarkastuksiin. Tarkastuksilla valvontajärjestelmät seuraavat laitteiden ja järjestelmien toimintaa. Tarkastukset ovat suoritettavien komentojen ja niihin liittyvien asetuksien yhdistelmä. Tarkastuksien asetuksissa määritellään yleensä ainakin tarkastuksen nimi, suoritustiheys, suoritettava komento ja sen argumentit. Komennoksi määritellään lähes poikkeuksetta jokin valvontaskripti. Tarkastuksille voidaan määritellä myös useita muita parametreja, jotka riippuvat käytetystä järjestelmästä. Jos tarkastukseen määritetyn valvontaskriptin, tai komennon, tarkastama asia on kunnossa, tarkastusta pidetään onnistuneena. Muissa tapauksissa tarkastuksen sanotaan epäonnistuneen. Tarkastukset yhdistetään valvontajärjestelmissä laitteisiin, joiden kuuluu suorittaa niitä. Ryhmät ovat yksi yleinen tapa yhdistää laite ja tarkastus toisiinsa. Kaikki ryhmään kuuluvat laitteet suorittavat siihen liitettyjä tarkastuksia. Tarkastuksien komennon tuloksen muoto on valvontajärjestelmästä riippuvainen.

Valvontaskriptit ovat valvontajärjestelmän suorittamia ohjelmia tai skriptejä, joiden tarkoitus on tarkastaa jonkin asian tila. Valvontaskriptit hyödyntävät yleensä niille annettuja parametreja tilan tarkastamista varten. Esimerkiksi valvontaskriptillä voitaisiin tarkastaa, onko jokin prosessi käynnissä. Skripti voisi ottaa parametreina vastaan prosessin nimen. Suorituksen loputtua skriptin tulos palautetaan valvontapalvelimelle, jossa se käsitellään. Jos skripti löytää annetulla nimellä prosessin, palvelin merkkaa tarkastuksen onnistu-

neen. Muussa tapauksessa palvelin merkkaa tarkastuksen epäonnistuneen. Valvontaskriptit voidaan yleensä jakaa tavallisiin valvontaskripteihin ja metriikankeräysskripteihin. Osa järjestelmistä ei erottele valvontaskriptejä eri kategorioihin.

Tavalliset valvontaskriptit tarkistavat jonkin asian tilan ja palauttavat tuloksen palvelimelle. Palvelin käyttää palautettua tulosta tarkastuksen tilan määrittämiseen. Tiloja on suurimmassa osassa järjestelmiä neljä: ok, varoitus, kriittinen ja tuntematon. Metriikankeräysskriptit mittaavat jotakin asiaa ja palauttavat kirjanpitoa varten yhden tai useamman luvun, joka kuvaa mitattua asiaa. Esimerkiksi metriikankeräysskripti voisi mitata kovalevyjen käyttöastetta ja palauttaa palvelimelle haluttujen tiedostojärjestelmien vapaan tilan määrän prosentteina ja tavuina. Kerättyä tietoa voidaan hyödyntää muun muassa tarvittavien resurssien ennakoimiseen. Järjestelmästä riippuen myös metriikankeräysskriptit voivat aiheuttaa hälytyksiä.

Monitoroitavien laitteiden kirjanpito valvontapalvelimella hoituu joko staattisesti tai dynaamisella rekisteröinnillä. Staattinen kirjanpito vaati järjestelmän ylläpitäjää lisäämään ja poistamaan laitteet järjestelmään manuaalisesti konfiguraatiotiedostoja muokaten. Dynaaminen rekisteröiminen mahdollistaa laitteiden lisäämisen ja poistamisen valvontajärjestelmästä sen ollessa päällä. Dynaamisen rekisteröimisen ansiosta mm. klustereihin voidaan lisätä, tai niistä voidaan poistaa, uusia palvelimia ilman konfiguraatioiden muuttamista. Nykyään dynaaminen rekisteröityminen on yleisempi tapa hoitaa kirjanpito. Kirjanpidon avulla järjestelmät pitävät yllä tietoa monitoroitavien laitteiden tiloista, tapahtumista ja muista tiedoista. Muita tietoja voisi olla esim. järjestelmän vastuuhenkilöt ja laitteen teknisentuen yhteystiedot. Näitä tietoja voidaan hyödyntää mm. järjestelmän lähettämissä hälytyksissä tai ne voidaan näyttää esimerkiksi valvontajärjestelmään kytketyssä web-käyttöliittymässä.

Valvonta-agentit ovat tietokoneille asennettuja ohjelmia, jotka vastaanottavat ja välittävät tietoa valvontapalvelimelle. Agenttien pääsääntöisinä tehtävinä on suorittaa palvelimen pyytämät valvontaskriptit ja lähettää skriptien tulokset palvelimelle käsiteltäviksi. Lisäksi agentteihin on usein sisäänrakennettu jonkinlainen keepalive- tai heartbeat-toiminto. Keepalive-toiminnon tarkoitus on ilmoittaa valvontapalvelimelle ajoittain agentin olevan vielä toimintakykyinen. Jos valvontapalvelin ei saa keepalive-viestejä agentilta tarpeeksi usein, agentti rekisteröidään toimintakyvyttömäksi. Toimintakyvyttömistä agenteista voidaan hälyttää ylläpitoa.

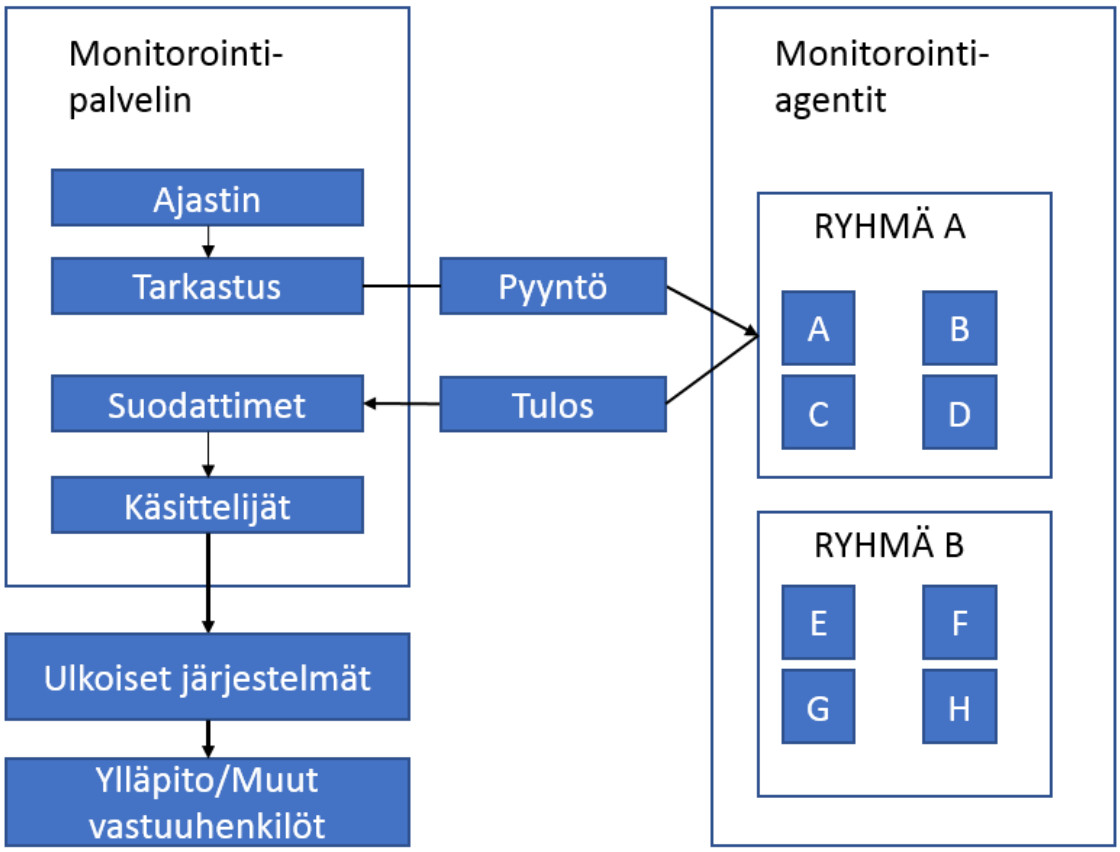
Valvontajärjestelmästä ja sen konfiguraatiosta riippuen palvelinkomponentti, valvonta-agentti, tai molemmat ajastavat valvontaskriptien suorittamista. Valvontajärjestelmät yleensä ajavat valvontaskriptejä kiinteiden aikavälien välein. Osa valvontajärjestelmistä tukee muitakin ajastustapoja, kuten ajamista vain määritettyyn kellon aikaan. Skriptien ajaminen voidaan suorittaa usein agentillisesti tai agentittomasti.

Skriptin suorittaminen agentilla tarkoittaa sitä, että valvonta-agentti suorittaa skriptin, jolla tarkistetaan agenttia ajavan laitteen tilaa. Agentittomalla ajamisella agentti suorittaa tarkastuksen, jolla tarkastetaan jonkin toisen laitteen tila ilman, että tarkastettavalle laitteelle on asennettu agenttia [3]. Agentittoman tarkastuksen tulos kirjataan järjestelmään niin kuin sen olisi suorittanut tarkastuksen kohteena ollut agentiton laite. Esimerkiksi moniin IoT- ja verkkolaitteisiin ei voida asentaa valvonta-agentteja, joten niiden valvonta täytyy suorittaa agentittomasti. Valvontajärjestelmästä ja laitteista riippuen agentittoman valvonnan toteutukset saattavat poiketa, mutta yleinen tapa agentittomaan valvontaan on hyödyntää erikseen agentittomia tarkastuksia varten kirjoitettuja valvontaskriptejä. Esimerkiksi joidenkin verkkolaitteiden valvontain voitaisiin käyttää skriptejä, jotka hyödyntävät SSH:ta komentojen ajamiseen etänä. Vaihtoehtoisesti voidaan käyttää laitteiden API:ja, jos laiteet tarjoavat sellaisen.

Osa valvontajärjestelmistä tukee passiivisia tarkistuksia. Passiiviset tarkistukset ovat tarkastuksia, jotka ovat valvontajärjestelmän näkökulmasta jonkin ulkoisen prosessin ajamia. Tulokset syötetään valvontajärjestelmälle sen tarjoamaa API:a hyödyntäen. Valvontajärjestelmän ajastamia tarkastuksia kutsutaan aktiivisiksi tarkastuksiksi. [4.]

Valvontajärjestelmät hyödyntävät tapahtumakäsittelijöitä ja suodattimia järjestelmän tuottamien tapahtumien käsittelyssä. Tapahtumankäsittelijän tehtävänä on suorittaa jokin toiminto hyödyntäen tapahtuman tietoja. Tapahtuman tiedot voidaan esim. lähettää järjestelmän vastuuhenkilöille yhdellä käsittelijällä ja toisella lähettää tekstiviesti päivystäjälle. Suodattimien tehtävänä on suodattaa käsittelijöille vietävät tapahtumat esim. tapahtuman tietojen ja valvontajärjestelmästä löytyvien historiallisten tietojen perusteella. Suodatin voisi suodattaa pois kaikki kehitysympäristöissä tapahtuneet tapahtumat, jotka ovat menossa tekstiviestejä lähettävälle tapahtumankäsittelijälle. Eri järjestelmät käyttävät eri nimiä samoille asioille, ja implementaatiot vaihtelevat. Osa järjestelmistä toteuttaa käsittelijät ja suodattimet erillisinä järjestelmän osina ja osassa ne on yhdistetty yhdeksi komponentiksi. [5; 6]

Valvontajärjestelmiin on yleensä saatavilla jonkinlainen kojelauta (eng. dashboard) monitoroitavien laitteiden tilojen seuraamista varten. Kojelaudat tyypillisesti näyttävät vähintään listat aktiivisista hälytyksistä, monitoroitavista laitteista, laitteilla ajettavista tarkastuksista ja kaikista järjestelmässä olevista tarkastuksista. Osa kojelaudoista mahdollistaa myös pääsyn järjestelmän muihin ominaisuuksiin kuten uusien laitteiden lisäämiseen järjestelmään.



Kuva 1. Ryhmälle A ajastetun tarkastuksen kulku

Kuvassa 1 on kuvattu yleistetysti valvontajärjestelmäien tarkastuksien kulku. Kuvan järjestelmässä palvelin ajastaa tarkastukset. Ajastin käynnistää tarkastuksia määräajoin, jotka ohjataan niitä suorittaville agenteille. Kuvassa tarkastus on ohjattu ryhmäperusteisesti ryhmälle A. Ryhmään kuuluvat agentit vastaanottavat tarkastuspyynnön ja suorittavat siihen konfiguroidun komennon (yleensä valvontaskriptin). Komennon tulos palautetaan valvontapalvelimelle. Tuloksen muoto ja sisältö riippuvat valvontajärjestelmästä. Tulokset suodatetaan erilaisilla perusteilla ennen kuin ne ohjataan tarkastukseen liitetyille tapahtumankäsittelijöille. Tapahtumankäsittelijät tekevät jotain hyödyllistä tarkastuksen tuloksia hyödyntäen. Tyypillisesti ne hyödyntävät ulkoisia järjestelmiä, joiden

avulla epäonnistuneista tarkastuksista lähetetään hälytyksiä ylläpidolle ja muille vastuuhenkilöille. Ulkoinen järjestelmä voi olla esimerkiksi sähköpostipalvelin. Lopuksi viesti päätyy vastuuhenkilöille, joiden tehtäväksi jää päättää, miten hälytyksen aiheuttanut ongelma ratkaistaan.

2.1 Hälytysten hallinta

Koska hälyttäminen on yksi valvontajärjestelmän tärkeimmistä tehtävistä, hälytysten konfigurointi ja hälytyksiin liittyvät ominaisuudet ovat tärkeässä asemassa. Hälytysten määrän hallitsemista varten järjestelmistä löytyy useita tapoja hallita järjestelmän tuottamien hälytysten määrää.

Hälytysten määrää voidaan hallita jossain määrin pelkästään konfiguroimalla tarkastuksien perusasetukset järkevästi. Yksi suoraan hälytysten määrään vaikuttava tekijä on tarkastuksen ajotiheys. Järjestelmät normaalisti lähettävät hälytyksiä vain tarkastuksen ajamisen jälkeen. Jos tarkastus ajetaan viiden sekunnin välein, hälytyksiä voi tulla tiheimmillään myös viiden sekunnin välein kyseiselle tarkastukselle. Esimerkiksi levytilan tarkastaminen joka minuutti ei välttämättä ole tarpeellista. Jos tila tarkastetaan vain kerran tunnissa, vuorokauden aikana hälytyksiä voi syntyä maksimissaan kaksikymmentäneljä. Ajotiheyden laskeminen voi kuitenkin vaikuttaa negatiivisesti monitoroinnin tarkkuuteen, koska nopeita muutoksia ei pystytä huomaamaan ja tapahtumahetken tarkkuus pienenee.

Tarkastuksien ajotiheyden lisäksi tarkastuksille voidaan määrittää hälytystiheys. Hälytystiheys kertoo järjestelmälle, kuinka usein samasta ongelmasta hälytetään. Jos edellisestä hälytyksestä on kulunut vähemmän aikaa kuin hälytystiheydeksi on määritetty, järjestelmä jättää hälytyksen lähettämättä. Hälytystiheyden säätäminen on yleensä parempi vaihtoehto hälytysten määrän hallintaa kuin ajotiheyden säätäminen.

Väärät positiiviset voivat aiheuttaa paljon vaivaa valvonnasta vastaavalle henkilöstölle. Väärillä positiivisilla tässä yhteydessä tarkoitetaan hälytyksiä, jotka ovat monitoroinnista vastaavalle henkilökunnalle turhia. Tilanteet, jotka korjaantuvat itsekseen ennen kuin kukaan ehtisi reagoimaan hälytykseen, on yksi esimerkki vääristä positiivisista. Väärät positiiviset voivat helposti aiheuttaa suuren määrän turhia hälytyksiä. Turhien hälytysten määrän hallintaan löytyy eri järjestelmistä paljon eri ominaisuuksia.

Yksi yksinkertaisimmista tavoista välttää vääriä positiivisia on tarkastuksien epäonnistumismäärän asettaminen tarkastuksille. Osa tarkastuksista on luonteeltaan hieman epäluotettavia tai nopeasti itsensä korjaavia. Näissä tapauksissa hälytyksen lähettäminen vain yhden epäonnistuneen tarkastuksen jälkeen ei välttämättä ole järkevää. Esimerkiksi kasvanut www-sivun vasteaika ei välttämättä johdu oikeista ongelmista. Hitaus voi johtua esim. palvelimen uudelleen käynnistämisestä, jolloin palvelin ei ole ladannut kaikkea oleellista välimuistiin. Jos hälytykseen vaaditaan useampi epäonnistunut tarkastus, tiedetään, ettei hälytys johdu esimerkiksi ohimenevästä ongelmasta.

Huoltokatkojen aikana kaikki järjestelmät eivät välttämättä toimi niin kuin niiden pitäisi, joten huoltokatkojen aikana hälytyksiä saattaa syntyä turhaan. Useat järjestelmät tukevatkin huoltoikkunoiden asettamista. Huoltoikkunan aikana huollon piiriin kuuluvista laitteista, tai vain osasta niillä ajettavista tarkastuksista, ei lähetetä hälytyksiä. Huoltoikkunoiden käytöllä voidaan välttää suuri määrä turhia hälytyksiä, jotka johtuvat huoltotoimenpiteistä.

Tarkastuksien välisillä riippuvuuksilla voidaan rajoittaa toisistaan riippuvien tarkastuksien aiheuttamia turhia hälytyksiä. Tarkastukset, jotka eivät voi onnistua toisen tarkastuksen jo epäonnistuttua, on hyvä konfiguroida toisistaan riippuvaisiksi. Riippuvuuksilla voidaan parhaillaan pudottaa turhien hälytysten määrää valtavasti. Esimerkiksi verkkolaitteiden vikaantuminen voi pudottaa useita verkkolaitteita verkosta. Jos vikaantuneen laitteen toimivuus on määritetty sen takana olevien laitteiden tarkistuksien riippuvuudeksi, järjestelmä ei hälytä erikseen jokaisesta verkosta pudonneesta laitteesta, vaan vain vikaantuneesta laitteesta. Näin järjestelmä lähettää vain yhden hälytyksen, joten oleellinen tieto ei häviä suureen hälytysmäärään.

Räpsyminen (eng. flapping) valvontajärjestelmissä on ongelma, joka syntyy, kun tarkastus epäonnistuu ja onnistuu toistuvasti. Koska räpsyminen voi synnyttää suuren määrän hälytyksiä, monissa järjestelmissä on tuki räpsymisen tunnistamiselle. Räpsyminen johtuu usein huonosti konfiguroiduista tarkastuksista, mutta voi johtua myös muista syistä. Esimerkiksi heikosti toimiva verkkoyhteys voi aiheuttaa räpsymistä joidenkin tarkastuksien kanssa. Räpsymisen tunnistusominaisuuksilla tarkastuksen tila pidetään keinotekoisesti samana hälytyksien lähettämisen kannalta, jolloin hälytyksiä ei lähetetä liikaa tarkastuksen räpsyessä.

Osa valvontajärjestelmistä tukee suodattimia. Suodattimet ovat ohjelmia, joiden tehtävänä on suodattaa hälytyksiä. Suodattimien tarkoitus on mahdollista loppukäyttäjiä luomaan uusia tapoja suodattaa hälytyksiä, koska sisäänrakennetut ominaisuudet eivät välttämättä ole tarpeeksi joustavia kaikkiin käyttötapauksiin. Esimerkiksi arkipyhinä syntyneet hälytykset voitaisiin suodattaa omalla suodattimella.

Valvontajärjestelmiin voidaan kytkeä myös erillinen hälytysten hallintajärjestelmä. Hälytysten hallintajärjestelmällä voidaan paikata valvontajärjestelmän puutteita ja helpottaa hälytysten hallintaa. Lisäksi erillistä järjestelmää käyttämällä voidaan myös paremmin jakaa järjestelmien ja niistä vastaavien henkilöiden vastuut.

2.2 Valvontajärjestelmän valitseminen ja käyttöönotto

2.2.1 Valinta

Valvontajärjestelmiä ja -työkaluja lähdettiin evaluoimaan tavoitekeskeisesti. Kaksi korkeimman tason tavoitetta oli pystyä valvomaan, että kriittiset laitteet ovat päällä ja kiinni verkossa sekä että tarvittavat palveluprosessit ovat päällä. Valvontajärjestelmän tulisi myös pystyä automaattisesti lähettämään hälytyksiä havaituista ongelmista järjestelmien vastuuhenkilöille yrityksen käyttämiä kommunikointityökaluja käyttäen. Järjestelmän toivottiin myös olevan helposti konfiguroitavissa ja sivuttaissuunnassa skaalautuvissa. (Sivuttaissuunnassa skaalautumisella tarkoitetaan kapasiteetin kasvattamista lisäämällä lisää tietokoneita järjestelmään. Pystysuuntainen skaalautuminen vastakohtaisesti tarkoittaa kapasiteetin kasvattamista lisäämällä esimerkiksi muistia ja parempi suoritin tietokoneeseen.). Lisäksi haluttiin, että järjestelmällä on helppo toteuttaa agentitonta valvontaa. Käytännössä lähestulkoon kaikki modernit valvontajärjestelmät täyttävät tärkeimmät vaatimukset. Järjestelmien suosion ja käyttömäärään arvioimiseen käytettiin Google Trends -palvelua.

Lupaaville järjestelmille tehtiin SWOT-analyysit ja niitä vertailtiin keskenään. Järjestelmien vahvuuksiin listattiin asioita kuten lisäominaisuudet sekä konsultoinnin ja kirjallisuuden saatavuus. Riskeinä järjestelmille listattiin muun muassa järjestelmän uutuus. Heikkouksina erityisesti vanhoilla järjestelmillä oli konfiguroinnin hankaluus ja joustamattomuus. Mahdollisuuksiin listattiin mm. mahdollisuus vaikuttaa järjestelmän kehitykseen.

Suurimmat erot järjestelmissä löytyivät niiden ylimääräisissä toiminnallisuuksissa. Vertailussa olleet kaupalliset järjestelmät tarjosivat poikkeuksetta web-käyttöliittymän, tai valvontakojelaudan (eng. monitoring dashboard), järjestelmää varten. Kaikille ilmaisille järjestelmille ei löytynyt virallisia web-käyttöliittymiä, mutta niihin löytyi usein yksi tai useampi kolmannen osapuolen kehittämä web-käyttöliittymä. Muita eroja järjestelmien välillä oli esim. virallisesti tuettujen valvontaskriptien ja lisäosien määrissä, käyttäjienhallintaominaisuuksissa, automaattisessa ongelmien korjaamisessa (eng. auto-remediation, self-healing), järjestelmien automatisoidun käyttöönoton helppoudessa jne. Kaupallisissa järjestelmissä toiminnallisuuksia oli yleensä enemmän ilmaisiin verrattuna. Lisäksi kaupallisten järjestelmien lisensseihin kuului usein käytön tuki.

Muutaman jatkoon valitun järjestelmän testaamiseen ja vertailuun varattiin aikaa noin yksi työviikko. Viikon aikana pyrittiin pystyttämään jokaiselle järjestelmälle demoympäristö ja tutustumaan järjestelmien dokumentaatioihin. Viikon aikana testattiin mm. Nagios Corea, Icinga2:ta, Zabbixia ja Sensua.

Nagios Core on mahdollisesti kaikista tunnetuin avoimen lähdekoodin valvontajärjestelmä. Järjestelmän nimi oli alun perin NetSaint, joka myöhemmin vaihdettiin Nagiokseksi. Kun Nagioksen maksullinen versio julkaistiin, muutettiin maksuttoman version nimeksi Nagios Core [7]. Nagios Core suorittaa valvonnan vain paikallisesti vakiona, mutta tarkastuksia ajetaan kuitenkin käytännössä etänä jonkin agenttilisäosan avulla. Tunnetuin agentti Nagios Corelle on NRPE (Nagios Remote Plugin Executor). NRPE on ollut olemassa Nagios Coren julkaisusta lähtien. Yksi Nagioksen parhaista puolista on sen valvontaskriptien API. Sen yksinkertainen ja tehokas API mahdollistaa helpon skriptien kehittämisen järjestelmään millä tahansa ohjelmointikielellä. Nagioksen suosion ja sen API:n helppokäyttöisyyden takia Nagiokselle on saatavissa tuhansia valvontaskriptejä Nagios Exchangesta [8]. Nagios Exchange on portaali, josta löytyy mm. Nagioksen valvontaskriptejä ja dokumentaatiota. Nagios Exchangestä löytyvillä skripteillä voidaan valvoa mm. käyttöjärjestelmän toiminnan osa-alueita, palvelinohjelmien toimintaa ja ohjelmien lisenssien käyttöä. Nagioksen kaupallinen versio on Nagios XI. Kaupallisessa versiossa on kaikkien Nagios Coren ominaisuuksien lisäksi paljon ominaisuuksia, joita ei löydy Nagios Coresta.

Nagios Coresta on haaraantunut muutamia projekteja, joista ehkä tunnetuin on Icinga. Icinga pyrki korjaamaan joitakin Nagioksen puutteita. Icingassa järjestelmän arkkitehtuuri on uusittu. Yksi tärkeimmistä ominaisuuksista, jota Nagioksesta ei löytynyt haarautumisen aikaan, on yksi yhtenäinen API Icingaan, usean hajautetun sijasta. Icinga2:ssa,

Icingan toisessa versiossa järjestelmään lisättiin tuki klusteroinnille. Icinga käyttää Nagios Coren valvontaskriptien API:a kuten monet muutkin valvontajärjestelmät.

Zabbix oli ainut valvontajärjestelmä, jota harkittiin, jossa oli erilainen API valvontaskripteille. Zabbixille löytyy epävirallinen lisäosa, jolla Nagios-yhteensopivia skriptejä voitaisiin käyttää sen kanssa tarvittaessa. Kaikista vertailluissa olleista järjestelmistä Zabbix vaikutti yhtenäisimmältä. Zabbixissa on sisäänrakennettu tuki metriikkatiedon tallettamista ja esittämistä varten, joten erillisiä työkaluja ei ole pakko käyttää. Zabbxin käyttäjienhallinta oli ylivoimaisesti paras vertailussa olleista järjestelmistä. Muissa järjestelmissä käyttäjänhallintaominaisuudet olivat hyvin rajoittuneita, tai ne puuttuivat kokonaan.

Järjestelmää valittaessa huomiota kiinnitettiin erityisesti hälytysten määrän minimoimiseen. Liian suuri määrä hälytyksiä uuvuttaa monitoroinnista vastaavan henkilökunnan. Yrityksellä on käytössä satoja antureita, joiden toimintaa haluttiin monitoroida. Tämän takia tarkistuksien riippuvuudet, huoltotaukojen määrittäminen ja agentittoman valvonnan helppous vaikuttivat lopulliseen päätökseen paljon.

Monille ominaisuuksille, joita järjestelmät tarjosivat, ei nähty ainakaan vielä käyttöä. Palvelusopimuksen (SLA) mukaisen käytettävyyden ja MTBF:n tai MFOP:in automaattista laskentaa ei koettu tarpeellisiksi. MTBF (mean time between failures) on keskimääräinen aika vikaantumisten välillä. MFOP (maintenance-free operating period) on keskimääräinen aika, jonka laite toimii ilman huoltoa. Muillekin ei suoranaisesti valvontain liittyville ominaisuuksille annettiin vähemmän painoarvoa järjestelmää valittaessa.

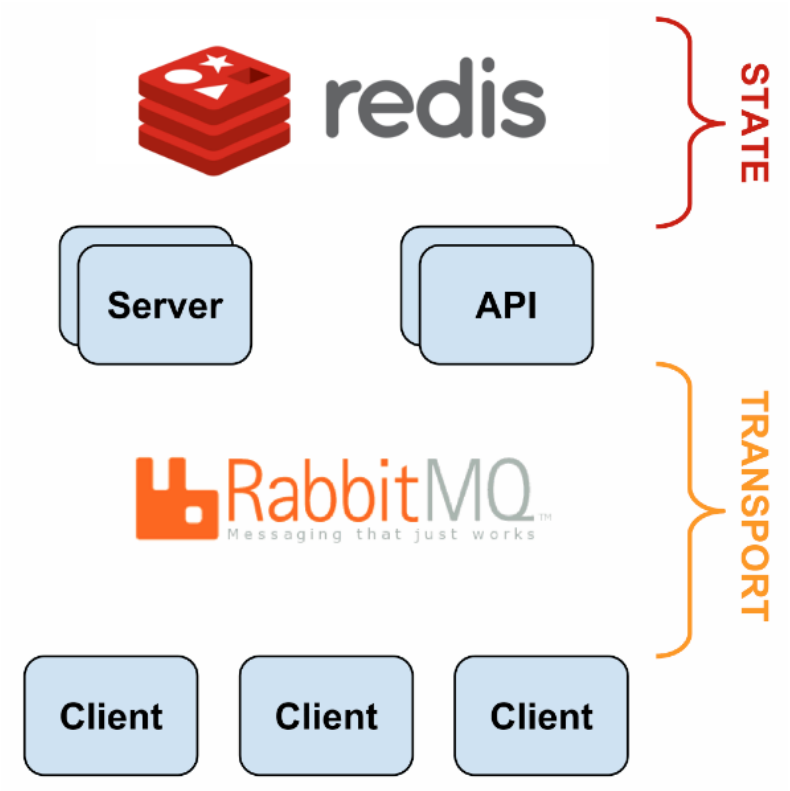
Metriikkatiedon kerääminen ei ollut osa valvonnan käyttöönoton ensimmäistä vaihetta, mutta metriikkatietojen keräämiseen kiinnitettiin kuitenkin huomiota. Lähes poikkeuksetta metriikkatietojen hyödyntäminen vaati aikasarjatietokannan (eng. timeseries database) ja jonkin työkalun datan hakemiseen ja esittämiseen. Järjestelmistä ainakin Zabbix tarjosi jonkin tasoisen metriikkatietojen tallettamisen ja hyödyntämisen perusominaisuutena.

Metriikkatiedon tallentamiseen ja analysoimiseen harkittiin Elasticsearchin käyttöä. Tyypillisesti metriikkatietoja talletetaan aikasarjatietokannoissa, mutta Elasticsearchia olisi voitu hyödyntää myös muihin yrityksen tarpeisiin. Elasticsearch on hajautettu haku- ja analytiikkamoottori. Elastisearchin toimivuuteen metriikkatiedon tallettamiseen tutustuttaessa saatiin selville, että CERN:in DIRAC:ssa Elasticsearcia käytetään tavallisten ai-

kasarjatietokantojen sijaan metriikkatietojen tallettamiseen [9]. DIRAC:ssa Elasticsearchia verrattiin InfluxDB:hen ja OpenTSDB:hen. Yrityksen käyttöön harkittiin samoja aikasarjatietokantoja, joten alustavasti pystyttiin päättämään, että seuraavassa vaiheessa voidaan aloittaa metriikkatiedon tallentamisen testaaminen Elastisearchiin.

Tavallisissa aikasarjatietokannoissa oli kuitenkin hyviä puolia, joita Elasticsearchista ei löytynyt. Hyvinä puolina on mm. automaattinen tarkkuuden pienentäminen (downsampling) vanhalle datalle ja datan säilytysajan määrittäminen. Molemmilla toiminnoilla voidaan vähentää levykapasiteetin tarvetta ja ylläpidon määrää. Vanhan datan tarkkuutta pienennettäessä data aggregoidaan esim. käyttäen usean datapisteen keskiarvoa, jolloin levykapasiteetin tarve pienenee. Datan säilytysajan ylittänyt data poistetaan järjestelmästä automaattisesti vähentäen ylläpidon ja levykapasiteetin tarvetta.

Noin kaksi viikkoa kestäneellä evaluointiajalla kävimme läpi usean järjestelmän ominaisuudet. Lupaaville järjestelmille pystytettiin pikaisesti demoympäristöt, joissa järjestelmien ominaisuuksiin ja helppokäyttöisyyteen tutustuttiin. Evaluointiajan lopussa päätettiin valita käyttöön Sensu.



Kuva 2. Sensun arkkitehtuuri

Sensu on melko uusi valvontajärjestelmä. Sen kehitys on aloitettu vuonna 2011 [10]. Sensua lähdettiin kehittämään vanhenevan Nagioksen korvaajaksi. Nagioksen valvontaskriptit ovat yhteensopivia Sensun kanssa. Automatisoinnin helppous on ollut aina yksi tärkeistä Sensun ominaisuuksista. Sensu on kehitetty silmällä pitäen mm. Ansiblea, Puppettia ja Cheffiä. Sensu hyödyntää Redis-tietokantaa järjestelmän tilan tallentamiseen ja RabbitMQ-viestijonopalvelinta (eng. broker) kommunikointiin palvelimen ja agenttien välillä (kuva 2). Sensun kaikki palvelinpuolen komponentit ovat klusteroitavissa [11; 12; 13].

Sensu valittiin mm. helppokäyttöisyyden, levittämisen ja konfiguroinnin automatisoinnin helppouden ja hyvän agentittoman monitoroinnin takia. Arkkitehtuurinsa pohjalta Sensu vaikutti helposti skaalattavissa olevalta. Järjestelmästä löytyivät kaikki tarvitsemamme ominaisuudet hälytysten hallintaa varten. Sensulle on helppo kirjoittaa omia suodattimia, joilla hälytystenhallintaa voidaan parantaa entisestään. Agentittomien tarkastuksien konfiguroiminen oli erittäin helppoa. Koska Sensisto Oy hyödyntää paljon verkkoon kytkettäviä antureita liiketoiminnassaan, joihin ei ole mahdollista asentaa minkään järjestelmän agentteja, agentittomien tarkastuksien helppous vaikutti valintaan positiivisesti. Valintaa tehdessä Sensusta puuttui vielä muutama ominaisuus, kuten cron-tyylinen tarkastuksien ajastus, jotka olivat tärkeitä. Ominaisuudet olivat kuitenkin jo kehityksessä, ja arvioitu julkaisupäivä oli lähellä.

Varavaihtoehdoksi valittiin Zabbix. Zabbixin levittäminen ja konfiguroiminen oli evaluoinnissa vaikeampaa kuin Sensun. Zabbix oli ominaisuuksiltaan kattavampi kuin Sensu, mutta kaikki oleelliset ominaisuudet löytyivät kuitenkin myös Sensusta. Zabbixin parhaita puolia Sensuun verrattuna olivat käyttäjänhallintaominaisuudet, joita käytännössä ei Sensussa ole. Zabbixilla on parempi tuki eri käyttöjärjestelmille kuin Sensulla. Paremmasta tuesta olisi ollut hyötyä Sensiston palomuureille, koska Sensua ei ole ainakaan vielä saatavilla niille.

2.2.2 Käyttöönotto

Sensun käyttöönotto tapahtui asteittain kahden kuukauden aikana. Käyttöönoton yhteydessä järjestelmän levitys ja konfigurointi haluttiin automatisoida ylläpidon helpottamiseksi. Automatisointiin käytettiin Ansiblea. Ansible on ohjelma, jolla voidaan automatisoida ohjelmien levittäminen palvelimille ja hallita järjestelmien konfiguraatioita. Auto-

matisointia varten luotiin muutama Ansible-playbook, joilla levitys ja konfigurointi hoidettiin. Ansible-playbookeissa määritellään suoritettavat toimenpiteet. Monitorointijärjestelmän levitys ja konfigurointi hoidettiin alusta pitäen Ansiblea hyödyntäen.

Valvontajärjestelmän tarkastukset jaettiin ryhmiin. Kaikki palvelimet kuuluvat vähintään perustarkastusryhmään, johon määritettiin tarkastukset, jotka jokaisen palvelimen on suoritettava tehtävästään riippumatta. Perustarkastusryhmässä tarkkaillaan mm. vapaata levykapasiteettia ja vapaata muistia. Muut ryhmät luotiin palvelimien tehtävien pohjilta. Noin puolet halutuista tarkastuksista pystyttiin suorittamaan käyttäen valvontajärjestelmän mukana tulleita tai Internetistä vapaasti saatavilla olevia valvontaskriptejä. Erityisesti agentitonta valvontaa varten jouduttiin kehittämään omia valvontaskriptejä. Yrityksessä agentitonta valvontaa käytetään lähinnä verkkoon kytkettyjen antureiden monitoroimiseen.

Hälytyksiä varten järjestelmään konfiguroitiin useita tapahtumankäsittelijöitä. Tapahtumankäsittelijät hyödyntävät kolmea eri tapahtumankäsittelyskriptiä. Tapahtumankäsittelijöillä hälytyksiä lähetetään sähköpostiin, yrityksen käytössä olevaan pikaviestiohjelmaan ja pysyvään tapahtumatietokantaan. Tapahtumankäsittelijöitä konfiguroitiin useita, jotta hälytysten määrä mm. pikaviestiohjelmaan pysyisi järkevänä. Lisäksi testausvaiheessa käytössä oli tapahtumankäsittelijä, jolla metriikkatietoja tallennettiin Elasticsearchiin. Elasticsearch-käsittelijä jouduttiin kehittämään erikseen, koska järjestelmään ei ollut valmista Elasticsearch-käsittelijää. Käsittelijä tehtiin valmiiksi seuraavaa vaihetta varten.

Käyttöönottoon sisältyi myös muita pienempiä asioita. Valvontajärjestelmää varten jouduttiin mm. luomaan SSL-sertifikaatteja, joita hyödynnettiin valvontapalvelimen ja -agenttien väliseen kommunikaatioon käytetyn RabbitMQ:n yhteyksien salaamiseen ja varmentamiseen. Suurin työ käyttöönotossa oli tutustua Nagioksen ohjeistukseen valvontaskriptien osalta ja toteuttaa valvontaskriptejä.

3 Nagioksen valvontaskriptien API

Sensu käyttää Nagios-yhteensopivaa valvontaskriptien API:a. API:ssa määritetään, missä muodossa valvontaskriptien tulokset tulee palauttaa valvontapalvelimelle.

Nagioksen valvontaskriptien API on hyvin yksinkertainen. Valvontajärjestelmä tarkistaa vain skriptin paluuarvon (eng. exit status, exit code tai return code). Paluukoodit 0, 1 ja 2 vastaavat tiloja ”OK”, "varoitus" ja "kriittinen". Kaikki muut paluukoodit tulkitaan "tuntematon"-tilaisiksi. Paluukoodia kolme tulisi kuitenkin käyttää, kun skriptistä halutaan palauttaa ”tuntematon”-tila. Skriptien STDOUT-tulosteet otetaan talteen ja palautetaan valvontajärjestelmälle paluukoodin kanssa. Järjestelmä voi tulosten käsittelyn jälkeen lähettää STDOUT-tulosteen esimerkiksi s-postilla palvelun ylläpidolle ja tapahtumatietokantaan. Nagioksen API vaatii, että skriptit tulostavat vähintään yhden rivin STDOUT:iin. Tulosteet ovat vain järjestelmän käyttäjiä varten. Nagioksen API ei ota STDERR-tulosteita talteen. Tämän takia STDERR-tulosteita voidaan käyttää esim. loppukäyttäjille esitettävien debug-viestien näyttämiseen. Koska vain paluukoodilla on väliä, voidaan valvontaskriptejä tehdä millä tahansa ohjelmointikielellä. Nagioksen virallisessa valvontaskriptipaketissa skriptit on kirjoitettu C- ja Perl-ohjelmointikielillä. [14.]

Nagioksen valvontaskriptien yksinkertainen API mahdollistaa periaatteessa minkä tahansa ohjelman käyttämisen tarkistuksien tekemiseen. Esimerkiksi grep-komennolla voitaisiin mahdollisesti tarkistaa järjestelmän lokeista jonkin ajastetun tehtävän onnistuminen. Monet valvontaskriptit käyttävätkin järjestelmän perustyökaluja avuksi tarkastuksissa. Yksinkertaisten skriptien tehtäviksi jääkin usein vain tulosteen muotoilu ja paluukoodin määrittäminen.

API:n yksinkertaisuutta voidaan pitää myös yhtenä sen heikkouksista. Valvontajärjestelmä ei voi tietää pelkän ohjelman paluukoodin perusteella, mitä järjestelmä teki. Virheellisiä paluukoodeja voi syntyä usealla tavalla. Virheellisten paluukoodien huomaaminen ja syiden löytäminen ovat helppoa suurimman osan ajasta, jos valvontaskriptien tulosteet saadaan talteen. Siitä huolimatta vääristä paluukoodeista voi koitua haittaa. Esimerkiksi väärien paluukoodien takia järjestelmä voi aiheuttaa turhia hälytyksiä tai ajaa turhaan automaattisia korjaustoimintoja.

Taulukko 1. Taulukko 1 Linux-paluukoodeja [15; 16]

Tapahtuma	**Paluu-koodi**	**Selite**
OOM-killer	137	Ohjelma tapettu muistin loppumisen vuoksi (SIGKILL)
Ohjelmaa ei löydy	127	Ohjelmaa ei löydy, virheellinen komento
SIGTERM	143	Ohjelma tapettu SIGTERM-signaalilla
SIGKILL	137	Ohjelma tapettu SIGKILL-signaalilla
Ei oikeutta suorittaa ohjelmaa	126	Execute-oikeus puuttuu

Taulukossa 1 on listattu muutamia esimerkkejä paluukoodeista, jotka käyttöjärjestelmä tai komentotulkki saattaa palauttaa ohjelman sijasta. Taulukon paluukoodit tulkitaan tuntematon tilaisiksi, joten ne ovat API:n mukaisia, koska keskeytyneen tarkastuksen tulos on tuntematon (ohjelmaa ei suoritettu kokonaisuudessaan). Paluukoodit eivät kuitenkaan ole valvontaskriptin palauttamia, joten niistä voi koitua pientä haittaa. Koodit 126 ja 127 aiheutuvat helposti, jos valvontaskriptejä levitettäessä jokin skripti unohtuu levittää tai käyttöoikeudet jäävät vääriksi. Käyttöjärjestelmän muistin loppuminen voi aiheuttaa sen, että OOM killer (Out-of-Memory killer) lopettaa jonkin ohjelman suorituksen SIGTERM- tai SIGKILL-signaalilla aiheuttaen paluukoodin 143 tai 137 [15; 16].

Osa vakiintuneista poikkeuksellisista paluukoodeista ei välttämättä kerro riittävällä tarkkuudella, mitä valvontaskriptiä ajettaessa tapahtui. OOM-killerin tappamaa ohjelmaa ei voi paluukoodin perusteella erottaa käyttäjän tappamasta ohjelmasta. Tieto OOM-killerin lopettamista ohjelmista löytyy kuitenkin yleensä järjestelmän lokeista, ja käyttäjien tappamien ohjelmien ei pitäisi olla ongelma.

Tuntematon-tilaisiksi merkattavat paluukoodit eivät ole ainoa ongelma. Paluukoodi yksi (varoitus) on yleinen paluukoodi, kun ohjelman suoritus loppuu virheeseen. Myös useat skriptitulkit antavat paluuarvon yksi niiden törmätessä virheeseen. Esimerkiksi puuttuvien kirjastojen käyttäminen skripteissä voi kaataa skriptin ja aiheuttaa paluukoodin yksi. Tämän lisäksi mikään ei estä ohjelmien kirjoittajia käyttämästä mielivaltaisia paluukoodeja ohjelmissaan. Monissa apuohjelmissa paluukoodeja hyödynnetäänkin ohjelman havaintojen välittämiseen käyttäjille. Esimerkiksi GNU diff:in paluukoodit 0 ja 1 kertovat, ovatko verrattavat tiedostot identtiset vai poikkeavatko ne toisistaan. Tämän takia valvontaskripteistä ei tule palauttaa suoraan toisilta ohjelmilta saatuja paluuarvoja.

Toinen ongelma ovat ohjelmat ja kirjastot, jotka tulostavat virheitä ja muita viestejä STDOUT-tiedostoon. Koska STDOUT-tuloste lähetetään valvontapalvelimelle, STDOUT:iin ei haluta turhia tulosteita. Joskus STDOUT joudutaankin ohjaamaan uudelleen jonnekin muualle, jotta turhat tulosteet eivät päädy palvelimelle. Kun turhia tulosteita aiheuttava osio skriptissä on ohitettu, voidaan STDOUT ohjata takaisin alkuperäiseen tiedostoon.

Monet API:n yksinkertaisuudesta johtuvat ongelmat ovat kuitenkin harvinaisia, jos järjestelmä on konfiguroitu oikein. Väärien hälytysten viesteistä pitäisi selvitä jollain tasolla ongelman syy, jolloin se voidaan korjata. Käytännössä vääriä hälytyksiä pitäisikin syntyä vain, jos skriptiä suorittavan tietokoneen konfiguraatiossa on ongelmia kuten puuttuva

kirjasto tai tietokoneen muisti on lopussa. Epätoivottujen paluukoodien pitäisi rajoittua hyvin konfiguroidussa järjestelmässä vain skriptistä ja konfiguraatioista riippumattomiin tekijöihin kuten OOM-killeriin.

Nagioksen virallinen ohjeistus neuvoo, että valvontaskriptien tulosteesta olisi hyvä selvitä monitoroitavan asian tila ja tarpeelliset yksityiskohdat tilasta. Tulosteelle suositellaan muotoa: ”TILA: lisätiedot”. Tuloste voidaan tarvittaessa jakaa useammalle riville, mutta se ei ole suositeltavaa. Jos tuloste jaetaan useammalle riville, tulosteen ensimmäisen rivin tulisi olla kaikista informatiivisin. Tulosteen rivipituuden olisi hyvä olla maksimissaan kahdeksankymmentä merkkiä, jotta tuloste on helposti luettavissa myös pieni resoluutioisilla laitteilla kuten palvelinhuoneen näytöiltä tai hakulaitteilta (eng. pager). Tuntematon-tilan käyttöä suositellaan, kun skriptille on annettu huonot parametrit tai skripti ei pysty suorittamaan tarkistustaan. Valvontaskriptien ei pidä kutsua muita ohjelmia käyttäen ohjelman nimeä, vaan ohjelman absoluuttista polkua. Absoluuttisen polun käyttäminen estää hakupolun kaappaamisen. Skriptien olisi myös hyvä välttää väliaikaisten tiedostojen luomista ja symbolisten linkkien käyttämistä. [17.]

Nagioksen virallisesta ohjeistuksesta löytyy myös paljon muita neuvoja. Osa neuvoista koskee vain Nagioksen kirjastoja hyödyntäviä valvontaskriptejä, mutta neuvoista löytyy myös muille järjestelmille hyödyllisiä ohjeita. Nagioksen ohjeistuksessa neuvotaan mm. ohjelmoimaan valvontaskripteihin aikakatkaisu, jotta suoritukseen ei kuulu aikaa enempää kuin halutaan. Nagioksen ohjeistus on hyvä paikka tutustua Nagioksen valvontaskriptien API:iin.

Työn aikana harkittiin myös Zabbixin käyttämistä yrityksen monitoroinnissa. Zabbixin API valvontaskripteille on selvästi erilainen ja ainakin osin miellyttävämpi kuin Nagioksen. Zabbix hyödyntää valvontaskriptin STDOUT- ja STDERR-tulosteita ja palvelimelle asetettuja raja-arvoja [18]. Nagios-yhteensopivissa järjestelmissä raja-arvot annetaan yleensä argumentteina valvontaskripteille. Zabbix ei muutamaa poikkeavaa versiota lukuun ottamatta välitä valvontaskriptin paluukoodista. Zabbix-palvelimelle määritetään, minkä tyyppisen tulosteen tarkastus tuottaa. Jos tulosteet ovat numeerisia, tarkastuksen tuloksia voidaan hyödyntää metriikoina. Zabbixin API:n suurin ero lienee se, että vain palvelin on tietoinen tarkastuksen onnistumisesta. Jos skriptin tuloste ei vastaa palvelimelle asetettuja ehtoja, tarkastuksen tulokseksi merkitään tukematon (unsupported).

4 Valvontaskriptit

Valvontaskriptit on hyvä suunnitella suorittamaan yksi tarkastus tehokkaasti ja varmasti. Valvontaskripti, joka toimii epävarmasti, voi olla jopa haitallinen valvonnan kannalta. Epävarmoja skriptejä käyttäviä tarkastuksia voi olla vaikea konfiguroida järkevästi hälytyksien kannalta. Pahimmillaan epävarmat skriptit tuottavat vääriä tuloksia, ja valvonnan hyöty katoaa.

Tarkkailijaefekti (eng. observer effect) ei normaalisti vaikuta nykytietokoneiden valvontaan [19]. Siitä huolimatta valvontaskriptien olisi hyvä olla mahdollisimman kevyitä. Tämä on erityisesti totta, jos yksi tietokone vastaa useiden muiden agentittomien laitteiden monitoroinnista. Jos valvontaskriptit ovat hitaita ajallisesti tai käyttävät paljon muistia, tämä saattaa vaikuttaa valvontaan tai tietokoneen muihin toimintoihin.

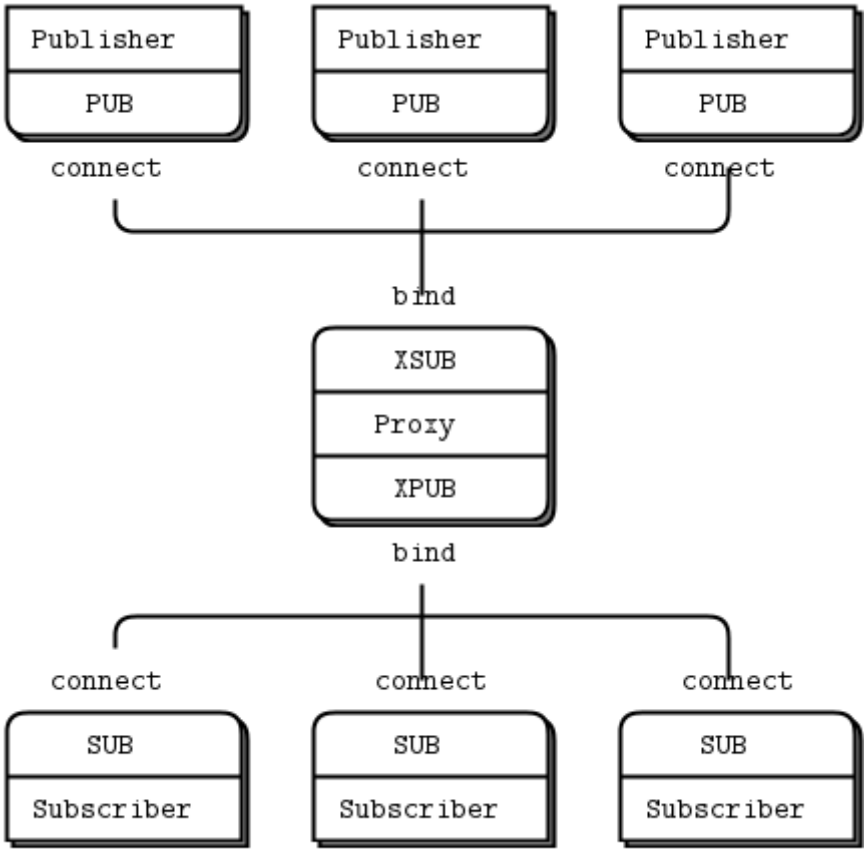
Valvontaskriptit (tarkastukset) on syytä suunnitella tarkistamaan järjestelmän toimintoja suoraan käyttäjän näkökulmasta silloin kuin mahdollista. Esimerkiksi valvontakameran toimivuutta ei välttämättä kannata tarkistaa vain pingillä, koska laiteen verkko-ominaisuudet saattavat toimia, vaikka kameralta ei saataisi kuvaa. Toiminnon tarkastaminen käyttäjän näkökulmasta ei välttämättä ole kuitenkaan aina mahdollista, jolloin joudutaan tyytymään muihin tarkastuksiin. Esimerkiksi salasanasuojattujen kameroiden monitoroiminen ei välttämättä onnistu esim. yrityksen tietoturvapolitiikasta johtuen.

Jos skripteistä kutsutaan muita ohjelmia, kutsuttujen ohjelmien tunteminen kunnolla on tärkeää. Osa ohjelmista ei käyttäydy nätisti Nagioksen valvontaskriptien API:n kanssa ja erot ohjelmien versioissa voivat muuttaa toimivan skriptin toimimattomaksi. Esimerkiksi jokin ohjelmalle annettavista vivuista on voinut poistua käytöstä, jolloin ohjelma voi lopettaa toimintansa virheeseen.

Kaikkia ohjelmia ja palveluita, joita halutaan valvoa, ei välttämättä pystytä valvomaan järkevästi esimerkiksi lokeista tai ohjelman tarjoamasta API:sta. Näissä tapauksissa voi olla mahdollista kirjoittaa valvontaa varten apupalvelu, joka pitää kirjaa ohjelman tai palvelun tilasta epäsuorasti. Tämänkaltaisia apupalveluita ei välttämättä tarvita, jos ohjelmalta saadaan kerättyä haluttu tieto metriikoina talteen. Näissä tapauksissa apupalvelun sijasta tarkastuksilla tarkastetaan metriikkatietokannasta halutut arvot. Apupalveluiden käyttäminen monimutkaistaa järjestelmää ja lisää uuden mahdollisesti hajoavan komponentin järjestelmään.

4.1 Valvonnanavustuspalvelu

Eräs yrityksen ohjelma välittää antureista luettua dataa muille ohjelmille. Ohjelma hoitaa datan siirron käyttäen ZMQ-viestijonokirjastoa (Zero Message Queue). ZMQ on kevyt viestijonokirjasto. Yksi sen vahvuuksista on mahdollisuus toimia ilman erillistä viestienvälityspalvelinta (eng. message broker). Sen avulla on helppo kirjoittaa hajautettuja järjestelmiä. ZMQ tukee useita eri tapoja viestien välitykseen kuten julkaisijatilaaja-mallia. Verkkoyhteyden hallinta on piilotettu loppukäyttäjältä, joten käyttäjän ei tarvitse huolehtia yhteyden hallintaan liittyvistä ongelmista.



Kuva 3. Julkaisija-tilaaja-verkkoarkkitehtuuri välittäjällä

Yrityksen anturidatan välitysohjelma hyödyntää ZMQ:n PUB-SUB-verkkoarkkitehtuuria välittäjällä (kuva 3) tiedonsiirtoon usean erilaisen anturityypin ja niitä hyödyntävien ohjelmien välillä. Verkossa anturit ovat julkaisijoita (PUB) ja dataa hyödyntävät ohjelmat tilaajia (SUB). Julkaisija-tilaaja-verkossa julkaisijat lähettävät viestejä, joihin on liitetty aihe. Tilaajat päättävät, minkä aiheisia viestejä ne haluavat vastaanottaa.

Eräälle anturityypille haluttiin luoda valvontaskripti, jolla valvotaan keskimääräistä datan prosessointiaikaa ja keskimääräistä viestien saapumisväliä. Koska välitysohjelma välittää erilaisten anturien tietoja, ja niiden valvonnan tarpeet vaihtelevat, valvontarajapinnan toteuttaminen ei ollut järkevää välitysohjelmaan (kuvassa 3 Proxy). Lisäksi rajapinnan toteuttaminen välitysohjelmaan pahimmillaan heikentäisi sen suorituskykyä ja toimintavakautta. Kaikille antureille ei myöskään tarvittu valvontaa datanvälitystasolla.

Koska datan välitysohjelmaan ei pystytty järkevästi toteuttamaan API:a valvontaa varten, päätettiin valvontaa varten kirjoittaa erillinen apupalvelu. Apupalvelu on viestienvälityksen näkökulmasta kuin mikä tahansa muu tilaaja verkossa. Apupalvelun tehtävänä on pitää kirjaa jokaiselta tietyntyyppiseltä anturilta viimeksi saadun viestin lähetysajoista ja prosessointiin kuluneesta ajasta. Prosessointiin kulunut aika on yksi viestin sisältämistä tiedoista. Anturityyppi valitaan tilaamalla viestit, joiden aihe vastaa haluttua anturityyppiä. Apupalvelu tarjosi REST-API:n tietojen lukemista varten.

```
import argparse
import json
import time
import requests

def main(url, wlimit, climit, request_timeout):
    try:
        r = requests.get(url, timeout=request_timeout)
    except requests.exceptions.Timeout:
        print("UNKOWN: Request timed out. (Timeout: {})".for-
mat(request_timeout))
        exit(3)
    data = json.loads(r.content.decode("utf-8"))
    now = time.time()
    diff = now - data["last_ts"]
    if diff >= climit:
        print("CRITICAL: {} seconds since last message".for-
mat(diff))
        exit(2)
    elif diff >= wlimit:
        print("WARNING: {} seconds since last message".for-
mat(diff))
        exit(1)
    print("OK")
    exit(0)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("url")
    parser.add_argument("-w", "--wlimit", type=int, default=30)
    parser.add_argument("-c", "--climit", type=int, default=60)
    parser.add_argument("--request-timeout", type=int, default=5)
    args = parser.parse_args()
    try:
        main(args.url, args.wlimit, args.climit, args.re-
quest_timeout)
```

```
    except Exception as e:
        print("UNKOWN: An error occurred during check execu-
tion\n{}".format(e))
        exit(3)
```

Esimerkkikoodi 1. Saapumisajan tarkastusskripti

Valvonnan avustuspalvelulta voidaan tarkastaa valvontaskriptillä (esimerkkikoodi 1), onko viimeisimmän viestin saapumisesta kulunut liian kauan aikaa. Skripti palauttaa vakiona OK:n, jos viimeisin viesti on saatu 30 s sisään. Yli 30 s viimeisestä viestistä palauttaa varoituksen (paluukoodi 1) ja yli minuutti palauttaa kriittisen (paluukoodi 2). Virheiden sattuessa ohjelma palauttaa tuntemattoman (paluukoodi 3). Skripti vaatii toimiakseen parametrina osoitteen, josta anturin tiedot haetaan. Argumentteina skriptille voidaan antaa raja-arvot varoitukselle ja kriittiselle tilalle sekunneissa sekä aikaraja yhteyden muodostumiselle.

Jos viimeksi saapuneiden viestien ajat olisivat saatavissa suoraan viestien välityspalvelusta, apupalvelua ei tarvittaisiin ja vältyttäisiin sen tuomalta monimutkaisuudelta. Vaihtoehtoisesti viestien tiedot voitaisiin tallentaa välitysohjelmasta esim. aikasarjatietokantaan ja kysellä tietoja sieltä. Käytännössä tietokantaan tallentamisen ja tietokannan ylläpidon hyödyt verrattuna apupalveluun olisivat kuitenkin pienet. Mikäli yrityksellä olisi käytössä enemmän vastaavaa infrastruktuuria vaativia järjestelmiä, tietokantaan tallentaminen olisi parempi ratkaisu.

4.2 Suoritusympäristö

Yrityksen käyttöön erikseen kehitetyt valvontaskriptit kehitettiin pääsääntöisesti Python-ohjelmointikielellä. Python-ohjelmointikielellä tehtyjä ohjelmia voidaan ajaa eristetyissä Python-ympäristöissä Virtualenv-työkalun avulla. Eristettyjen ympäristöjen yksi tärkeimmistä hyödyistä on mahdollisuus käyttää kirjastojen eri versioita eri ohjelmien kanssa. Monitoroinnin kannalta tästä on erityisesti hyötyä, jos monitoroitavalla palvelimella on Python-ohjelmia, jotka käyttävät samoja kirjastoja kuin valvontaskriptit. Eristämällä skriptit toiseen ympäristöön voidaan varmistaa, että sekä skriptit että palvelimen ohjelmat käyttävät kirjastoista versioita, joilla ne on testattu. Eristettyjen ympäristöjen käyttäminen vaikeuttaa hieman valvontaskriptien konfigurointia, koska skriptit täytyy konfiguroida ajettaviksi eristetyn ympäristön Python-tulkilla.

Jos valvontaskriptit käyttävät keskenään epäyhteensopivia versioita eri kirjastoista, ei voida käyttää yhtä eristettyä ympäristöä valvontaskriptejä varten. Yhden ympäristön käyttäminen helpottaisi järjestelmän konfigurointia, koska kaikki skriptit voitaisiin konfiguroida käynnistettäviksi yhdellä tulkilla. Jos skriptit käyttävät keskenään ristiriitaisia versioita, joudutaan ainakin osalle skripteistä luomaan omat eristetyt ympäristöt ja konfiguroimaan tarkastukset ajettaviksi omissa ympäristöissään.

Yrityksen valvontaskriptit ajetaan ainakin toistaiseksi omissa eristetyissä ympäristöissään. Yrityksen käyttöön usean eristetyn ympäristön käyttö otettiin, koska automatisoidun ohjelmistojen levityksen ansiosta tästä ei koitunut liikaa ylimääräistä työtä. Ainoa oleellinen haitta usean eristetyn ympäristön käytössä onkin suuri määrä kopioita samoista kirjastoista. Yhden eristetyn ympäristön käyttöön voidaan tarvittaessa siirtyä helposti.

Suurin osa yrityksen käyttöön kirjoitetuista valvontaskripteistä liittyy antureilta saatavan datan valvontaan. Yrityksessä haluttiin valvoa antureiden ja niihin liittyvien järjestelmien toimintaa kokonaisuudessaan. Valvontaa varten kirjoitettiin skriptejä, joilla varmistetaan antureiden toimivuus, dataa esikäsittelystä vastaavien ohjelmien toimivuus, datan siirron onnistuminen varastoon, datan käsittelyn eräajojen onnistuminen ja datan näkyvyys loppukäyttäjille. Monitasoisella valvonnalla voidaan helpottaa ongelmatilanteiden selvittämistä.

Valvontaskriptien kirjoittaminen vaati yrityksen järjestelmiin tutustumista ja muutamien kirjastojen käytön opettelua. Antureiden toiminnan varmistamista varten luotiin skriptejä. Antureiden toimintaa testattiin lukemalla niiden mittaamia tietoja. Suurimmalta osalta antureista tämä onnistui käyttämällä niiden HTTP- rajapintoja.

Yritystä varten kehitetiin antureihin liittyvän valvonnan lisäksi muutamia muita valvontaskriptejä. Yksi tärkeimmistä skripteistä oli palvelimien päivityksiä seuraava valvontaskripti. Yrityksen käytössä olevat palvelimet päivitetään suurimmalta osalta automaattisesti, mutta muutamien pakettien versiot on lukittu. Valvontaskriptillä seurataan palvelimelle saatavilla olevien tietoturvapäivitysten määrää. Jos palvelimelle on saatavissa tietoturvapäivityksiä, joita ei ole vielä asennettu, skripti palauttaa kriittisen tilan palvelimelle. Ylläpito voi hälytyksen pohjalta asentaa päivityksen testausta varten ja kaiken toimiessa päivittää palvelimilla olevat versiot.

5 Prototyyppiosio

5.1 Humidorin valvonnan johdanto

Opinnäytetyön toisessa osassa kehitettiin tietokoneeseen kytkettävä anturi (vrt. verkkoon kytkettävä) ja testattiin sen valvontaa yritykselle valittua valvontajärjestelmää käyttäen. Yrityksellä on käytössään verkkoon kytkettyjä antureita, jotka kytkettiin valvontajärjestelmään opinnäytetyön aikana. Tämän toisen osan tarkoituksena oli testata, kannattaako erillisiä tietokoneeseen kytkettyjä antureita monitoroida suoraan valvontajärjestelmällä.

Toisen osa anturin aiheeksi valittiin etäluettava kosteus- ja lämpömittari minun toiveestani. Omat tavoitteeni kehitettävälle anturille oli tehdä ensimmäinen käytännöllinen elektroniikkaprojekti ja helpottaa sikareiden säilömistä. Anturi päätettiin tehdä hyödyntäen Arduino Unoa. Vaikka kosteus- ja lämpötila-anturille ei ole ainakaan toistaiseksi suoraa hyötyä yrityksen käytössä, haluttiin, että anturilla on kuitenkin jokin käytännön käyttötarkoitus. Sikareiden säilyttämisen valvonta sopi tähän vaatimukseen.

Sikareita säilytetään yleensä noin 65-70 %:n suhteellisessa kosteudessa ja noin 70 °F:n (21 °C) lämmössä. Kuiva sikari palaa liian kuumasti, nopeasti ja epätasaisesti. Liian kosteaa sikaria voi olla vaikea sytyttää ja sen polttaminen voi olla vaikeaa. Sikari on ns. tukossa. Sikareiden säilyttäminen liian kosteassa voi saada sikareiden käärelehden hajoamaan sikarin täytteen laajentuessa. Liika kosteus tarjoaa myös otolliset olosuhteet homeen kasvamiselle. Sikarin lehdissä saattaa myös piillä kuoriaisten munia, jotka voivat kuoriutua kosteissa ja lämpimissä olosuhteissa. [20.]

Tyypillisesti normaalin huoneiston suhteellinen kosteus olisi hyvä pitää noin 45%:n paikkeilla. Tyypillinen vaihteluväli on 35–50 %. Yli 50 %:n suhteellinen kosteus kasvattaa homeen kasvamisen mahdollisuutta. Lisäksi pölypunkit tarvitsevat kosteutta lisääntyäkseen, joten liiallinen kosteus ei ole tästäkään syystä hyvästä. [21.]

Koska tyypillinen huoneiston kosteus ei ole riittävä sikareille, sikareiden säilyttämiseen käytetään humidoreja. Humidorit ovat yleensä melko tiiviitä laatikoita tai kaappeja, joiden tehtävänä on mahdollistaa sikareiden säilyttäminen suopeissa olosuhteissa. Yleensä humidorit tehdään puusta, mutta myös muovisia humidoreja löytyy markkinoilta. Tyypillinen laatikkomallinen humidorin on suunniteltu säilömään noin 50 corona-kokoluokan sikaria.

Humidorin tilavuus ja siellä säilytettävien sikareiden määrä vaikuttaa tarvittavan kosteutuksen määrään. Humidoreita kosteutetaan esimerkiksi pitämällä pientä vesiastiaa humidorin sisällä tai kostuttimilla, jotka sisältävät esimerkiksi akryylipolymeeristä valmistettuja vesihelmiä.

Omistamani humidorin mukana tuli analoginen kosteusmittari, joka oli asennettuna humidorin etupaneeliin. Työpaikallani pystyin vertaamaan kosteusmittarini tarkkuutta entisen lämpö- ja kosteusherkän tutkimushuoneen kosteusmittariin. Oman kosteusmittarini arvot heittivät huoneen mittarin arvoista noin kymmenen prosenttiyksikköä. Koska halusin mittarin tarkkuuden olevan noin viiden prosenttiyksikön sisällä todellisesta kosteudesta, lähdin etsimään digitaalista ratkaisua.

Digitaalisia humidoriin tarkoitettuja kosteusmittareita löytyy helposti ulkomailta. Halvimmat mittareista olivat liian epätarkkoja, kookkaita tai saatavuus oli huono. Riittävän tarkat mittarit, joita oli saatavilla, maksoivat kymmeniä euroja. Digitaalisten mittareiden käyttö vaatii yleensä myös humidorin avaamista, kun kosteus halutaan lukea mittarilta. Vaihtoehtoisesti mittarin voisi upottaa humidoriin, mutta tämä vaatisi sopivan reiän leikkaamista humidoriin. Langattomasti etäluettavia mittareita ei löytynyt markkinoilta, joten päätin tehdä langattomasti luettavan mittarin prototyypin.

5.2 Arduino-prototyyppi

Prototyyppi päätettiin tehdä Arduino-alustalle, koska alustalle on nopea kehittää laajan kirjastovalikoiman ansiosta. Arduinolle kehittäessä ei tarvitse tutustua liiakseen prosessorin rekistereihin ja muihin ominaisuuksiin. Prototyyppi kehitettiin käyttäen Arduino Unoa. Jos olisin joutunut ostamaan kehityslaudan työtä varten, olisin käyttänyt jotakin muuta lautaa kuten Arduino MKRZeroa, koska Uno ei kokonsa puolesta mahdu täyden humidorin sisään.

Kosteus- ja lämpömittarin prototyypin haluttiin olevan langattomasti luettavissa. Lukuetäisyydeksi haluttiin vähintään kaksi metriä. Kehitystä ja käyttöä varten haluttiin, että laitteen tietoja voidaan lukea sekä Windows- että Linux-käyttöjärjestelmillä. Tiedonsiirto Arduinosta langattomalle moduulille täytyisi toimia käyttäen Arduino Unon sarjaportteja. Viestit laitteelta lähetettäisiin JSON-muodossa rivinvaihdolla erotettuina.

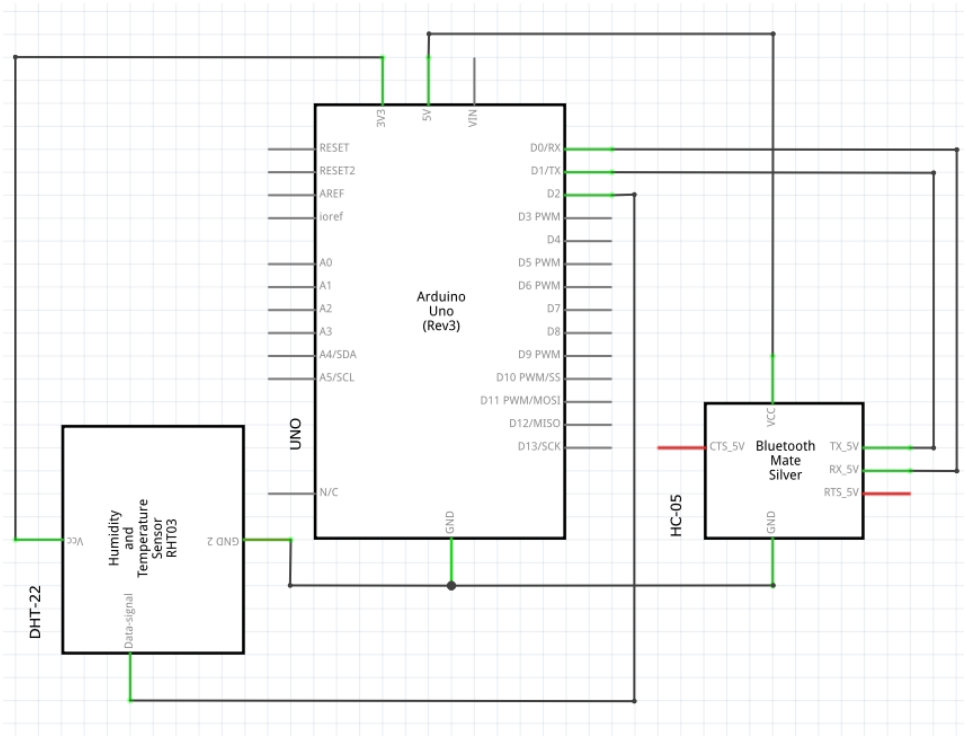
Kehittämistä varten tilattiin DHT22-kosteus- ja lämpötila-antureita (tunnetaan myös nimellä AM2302) ja HC-05 Bluetooth -moduuli. DHT22-anturin tarkkuus on kahdesta viiteen prosenttiyksikköön todellisesta ja lämpötila puolen celsiuksen sisällä. Anturi oli halpa ja riittävän tarkka käyttötarkoitukseen. Anturi käyttää omaa yhden linjan (single-wire) protokollaa tiedon siirtoon. Tiedot anturilta voidaan lukea kahden sekunnin välein. [22.]

Bluetooth-moduulin ainoat ostoperusteet olivat sen helppokäyttöisyys Arduinon kanssa ja sarjaporttien käyttö tiedonsiirrossa moduulin ja Arduinon välillä. Lisähyötynä moduuli yhdistää käynnistyessään automaattisesti viimeisimpään laitteeseen, johon se on ollut yhdistettynä. Haluttaessa olisi siis mahdollista kytkeä moduuli pois päältä, kun sitä ei tarvita, ja automaattisesti luoda yhteys uudestaan ilman mitään lisätoimenpiteitä käyttäjältä. [23.]

Prototyyppiä kehittäessä ei huolehdittu kokonaisuuden tai yksittäisten komponenttien sähkön kulutuksesta. Sähkön kulutuksen optimointi olisi vaatinut mm. Arduino Unon jännitteensäätimen korvaamista energiatehokkaammalla ratkaisulla. Arduinolle kirjoitettu ohjelma pyrittiin siitä huolimatta kirjoittamaan niin, että sähkön kulutus pysyisi mahdollisimman pienenä.

Prototyypin ohjelmointia varten asennettiin Arduino IDE ja tarvittavat ajurit. Kosteus- ja lämpötila-anturin lukemista varten ladattiin ja asennettiin DHT22-anturille kirjoitettu kirjasto DHTLib. Muut käytetyt kirjastot tulivat Arduino IDE:n mukana.

Kosteuden ja lämpötilan lukeminen laitteelta mahdollistettiin testikäytössä nappia painamalla. Alun perin oli tarkoitus käyttää kaksiosaista seitsemän segmentin led-näyttöä tiedon esittämiseen, mutta koekytkentälevyn pienen koon vuoksi ideasta luovuttiin. Nappi kytkettiin Arduinon porttiin D2. Portin ulkoiset keskeytykset kytkettiin päälle, ja porttiin liitettiin keskeytyksenkäsittelyrutiini (interrupt service routine) viestien lähettämistä varten. Arduino Unossa vain portteja D2 ja D3 voidaan käyttää ulkoisten keskeytysten aiheuttamiseen. Kun ensimmäiset viestit saatiin lähetettyä onnistuneesti, nappi poistettiin koekytkentälaudalta.



Kuva 4. Piirikaavio

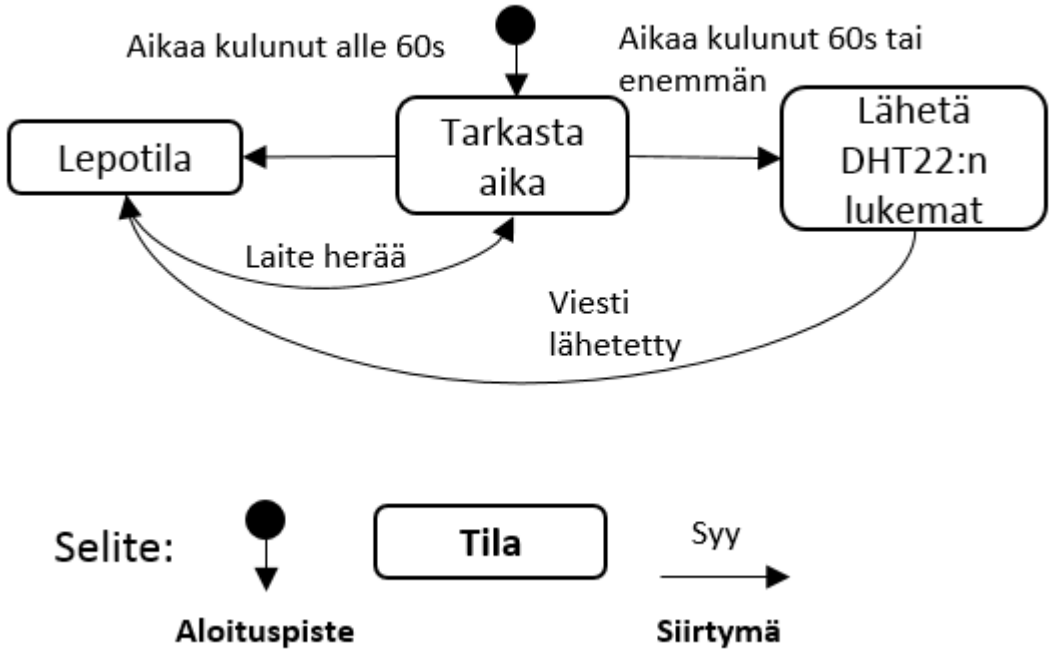
Valmiissa prototyypissä (kuva 4) käytetään Unon sarjaportteja (D0 ja D1) Bluetooth-moduulin kanssa kommunikoimiseen. Portilla D3 kommunikoidaan kosteus- ja lämpötila-anturin kanssa. HC-05 Bluetooth -moduuli näkyy Windows- ja Linux-käyttöjärjestelmille Bluetooth-sarjalaitteena. Prototyypin lähettämien viestien lukemista testattiin aluksi Arduino IDE:n SerialMonitor-sarjaliikenneterminaalia ja myöhemmin powershell-skripteillä.

Viestit lähetetään laitteelta JSON-muodossa Bluetoothin yli. Viestit on jaettu kahteen tyyppiin. Tyyppi merkitään JSON-viestiin merkkijonona type-avaimen alle. Sallitut arvot ovat ”msg” ja ”data”. Sanomat käyttävät tyyppiä ”msg”. Sanomilla lähetetään viestejä lokitusta varten. Sanomissa on viestin tyypin lisäksi sanoma merkkijonona msg-avaimen alla. Dataviesteissä tyyppi on ”data”. Lisäksi viestissä on lämpötila ja suhteellinen kosteus avaimien ”temp” ja ”rh” alla. Molemmat tiedot ovat liukulukuja maksimissaan kahden desimaalin tarkkuudella. Viestit erotetaan toisistaan rivinvaihdoilla.

5.2.1 Arduino-ohjelman läpikäynti

Arduino-ohjelman toiminta pohjautuu ajastimen, keskeytysten ja lepotilan hyödyntämiseen. Ajastinta hyödynnetään anturin lukemiseen jaksoittain ja Arduinon herättämiseen lepotilasta. Kun ajastimen aika täyttyy, se aiheuttaa keskeytyksen. Keskeytys herättää

laitteen lepotilasta ja ohjelman suoritus jatkuu ajastimen keskeytyksenkäsittelyrutiinista. Keskeytyksenkäsittelyrutiinin suorittamisen jälkeen ohjelma jatkaa suoritustaan lepotilaan siirtymistä seuraavasta komennosta. Lepotilaa ohjelmassa käytetään virrankulutuksen vähentämiseksi. Toimintalogiikka kuvattuna yksinkertaistettuna tilakoneena kuvassa 5. Ohjelma on kirjoitettu C++-ohjelmointikielellä.



Kuva 5. Arduino-ohjelman toimintalogiikka yksinkertaistettuna tilakoneena

```
#include <DHT.h>
#include <avr/sleep.h>
#include <avr/power.h>
#include <avr/wdt.h>

#define BAUD 9600
#define READ_SECONDS 60
#define DHT22_PIN 13

volatile int seconds = 0;
bool read_success = false;
dht DHT;
```

Esimerkkikoodi 2. Arduino-prototyypin kirjastot, makrot ja globaalit muuttujat

Arduino-prototyypin ohjelmakoodin alussa (esimerkkikoodi 2) sisällytetään ohjelman tarvitsemat otsikkotiedostot, määritellään makroilla vakiota ja määritetään ohjelman globaa-

lit muuttujat. DHT.h-otsikkotiedostossa määritellään dht-luokka, jota hyödynnetään ohjelmassa DHT22-anturin lukemiseen. Otsikkotiedostossa avr/sleep.h on määritelty prosessorin lepotilaan liittyviä asioita, avr/power.h-otsikkotiedostossa määritetään virranhallintaominaisuuksiin liittyviä asioita ja avr/wdt.h-otsikkotiedostossa Watchdog-ajastimeen liittyviä asioita. BAUD-makrolla määritetään Bluetooth-moduulin kanssa käytettävä signaalinopeus. READ_SECONDS-makrolla määritetään DHT22-anturin lukutiheys ja datan lähetystiheys. DHT22_PIN-makrolla määritetään Arduinon portti, johon DHT22-anturin datapinni on kytketty. Globaalilla seconds-muuttujalla seurataan kulunutta aikaa. Muuttuja on määritelty epävakaaksi (volatile). Epävakaaksi määrittäminen varmistaa sen, että muuttujan arvo luetaan aina muistista. Koska muuttujan arvoa muutetaan keskeytyksenkäsittelijässä, normaalin suorituspolun ulkopuolella, ohjelman täytyy hakea mahdollisesti muuttunut arvo aina muistista. Toisin sanoin rekistereissä mahdollisesti olevan arvon ei voi luottaa olevan oikea. Muuttujaa tarvitaan, jos halutaan ajastaa ajastimen maksimipituutta pidempiä aikoja. Boolean muuttujassa read_success pidetään tieto viimeisimmän DHT22-anturin lukukerran onnistumisesta. DHT-muuttujassa on dht-olio, jota käytetään DHT22-anturin lukemiseen.

```
ISR(WDT_vect) {
  seconds += 8;
}

void powerdown() {
  power_all_disable();
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  sleep_mode();
  sleep_disable();
}
```

Esimerkkikoodi 3. Arduino-prototyypin virransäästöön ja ajastukseen liittyvät funktiot

Watchdog-ajastimen keskeytys ja lepotilaan menemiseen käytettävä funktio on määritelty esimerkkikoodissa 3. Watchdog-ajastimen keskeytyksenkäsittelyrutiini määritetään käyttäen ISR-makroa. Makrolle annetaan argumenttina Watchdog-ajastimen keskeytysvektori. Makron jälkeen määritetään keskeytyksenkäsittelyrutiinin (funktion) runko. Prototyypissä keskeytyksenkäsittelyrutiinissa lisätään 8 globaaliin seconds-muuttujaan. 8 sekuntia on Watchdog-ajastimen pisin ajastusväli ja sitä päätettiin käyttää, koska mittausvälin haluttiin olevan vähintään minuutti. Keskeytyksen toinen tehtävä on herättää laite lepotilasta, jolloin ohjelmaan voi jatkaa suoritustaan. Funktiossa powerdown poistetaan lisäksi prosessorin turhat lisälaitteet pois käytöstä, määritetään lepotilan tyyppi ja siirrytään lepotilaan. Prosessori siirtyy lepotilaan sleep_mode-funktiokutsussa. Prosessori herää lepotilasta Watchdog-ajastimen aiheuttamaan keskeytykseen noin 8 sekunnin

välein, jonka jälkeen suoritetaan siihen liitetty keskeytyksenkäsittelyrutiini. Rutiinin suorituksen jälkeen ohjelma jatkaa suoritusta sleep_mode-funktiosta, josta se palaa sleep_disable-riville.

```
void read_sensor() {
  int chk = DHT.read22(DHT22_PIN);
  switch (chk)
  {
  case DHTLIB_OK:
    rh = DHT.humidity;
    t = DHT.temperature;
    read_success = true;
    break;
  case DHTLIB_ERROR_CHECKSUM:
  case DHTLIB_ERROR_TIMEOUT:
  default:
    read_success = false;
    break;
  }
}
```

Esimerkkikoodi 4. Arduino-prototyypin kosteus- ja lämpötila-anturin lukufunktio

DHT22-anturin lukeminen tapahtuu read_sensor-funktiossa (esimerkkikoodi 4). Funktiossa hyödynnetään globaalia dht-oliota DHT. Olion read22-metodi yrittää lukea kosteuden ja lämpötilan anturilta. Jos lukeminen onnistuu, metodi asettaa olion kosteus- ja lämpötilamuuttujiin luetut arvot ja palauttaa DHTLIB_OK:n (0). Epäonnistuttaessa palautetaan joko DHTLIB_ERROR_CHECKSUM (-1) tai DHTLIB_ERROR_ TIMEOUT (-2) riippuen epäonnistumisen syystä. Virheen sattuessa olion kosteus- ja lämpötilamuuttujien arvoiksi asetetaan DHTLIB_INVALID_VALUE (-999). Lukemisen onnistuessa globaalin read_success-muuttujan arvoksi asetetaan true; muuten arvoksi asetetaan false.

```
void send_values() {
  if (read_success) {
    Serial.print("{\"type\": \"data\", \"rh\": ");
    Serial.print(DHT.humidity, 1);
    Serial.print(", \"temp\": ");
    Serial.print(DHT.temperature, 1);
    Serial.println("}");
  } else {
    Serial.println("{\"type\": \"msg\", \"msg\": \"Read ERR\"}");
  }
  Serial.flush();
}
```

Esimerkkikoodi 5. Arduino-prototyypin datan lähetysfunktio

Arvot lähetetään Bluetooth-moduulille send_values-funktiossa (esimerkkikoodi 5). Funktiossa tarkastetaan aluksi, onnistuiko kosteuden lukeminen globaalista read_success-

muuttujasta. Arvot lähetetään sarjaliikennemoduulilla JSON-muodossa Bluetooth-moduulille, jos lukeminen onnistui. Muuten lähetetään virheviesti. Arvot viestiin saadaan globaalista dht-oliosta. Serial.flush-metodikutsulla varmistetaan, että viesti on siirtynyt Bluetooth-moduulille ennen funktiosta poistumista.

```
void setup() {
  cli();
  wdt_disable();
  // Enter Watchdog Configuration mode:
  WDTCSR |= (1<<WDCE) | (1<<WDE);
  //8s timeout, wdt isr on, reset off
  WDTCSR = (1<<WDIE)|(0<<WDE)|(1<<WDP0)|
           (0<<WDP1)|(0<<WDP2)|(1<<WDP3);
  sei();

  Serial.begin(BAUD);
  Serial.println("{\"type\": \"msg\", \"msg\": \"Staring up!\"}");
  Serial.flush();
  power_all_disable();
}
```

Esimerkkikoodi 6. Arduino-prototyypin setup-funktio

Setup-funktiossa (esimerkkikoodi 6) konfiguroidaan Watchdog-ajastin ja sarjaliikennemoduuli käyttöön. Watchdog-ajastin asetetaan keskeytystilaan ja ajastukseksi kahdeksan sekuntia. Keskeytystilassa ajastimen määräajan kuluttua ajetaan ajastimeen kytketty keskeytys. Arduino Unon prosessorin, ATmega328P:n, Watchdog-ajastin kytketään käyttöön kirjoittamalla muokkaamalla WDTCR-rekisterin bittejä. Yhdellä operaatiolla täytyy ensin asettaa WDTCR-rekisterin WDE- ja WDCE-biteille arvoksi yksi, ja seuraavan neljän kellosyklin aikana asettaa määräaika ja ajastimen toimintatila asettamalla WDTCR-rekisteriin halutut bitit yhdellä operaatiolla [24]. Setup-funktiossa poistetaan lisäksi kaikki ylimääräiset laitteet pois käytöstä. Arduino-ohjelmien suoritus alkaa setup-funktiosta, jota alusta kutsuu kerran käynnistyessään.

```
void loop() {
  if(seconds >= READ_SECONDS) {
    seconds -= READ_SECONDS;
    power_timer0_enable();
    power_usart0_enable();
    read_sensor();
    send_values(rh, t);
  }
  powerdown();
}
```

Esimerkkikoodi 7. Arduino-prototyypin pääsilmukka, loop-funktio

Prototyypin pääsilmukassa, loop-funktiossa (esimerkkikoodi 7), tarkastetaan globaalista seconds-muuttujasta, onko viimeisestä lähetyksestä kulunut READ_SECONDS-sekuntia tai enemmän. Jos aikaa on kulunut tarpeeksi, ohjelma lukee kosteuden ja lämpötilan DHT22-anturilta ja lähettää lukemat Bluetooth-moduulille send_values-funktiolla. Globaalista seconds-muuttujasta vähennetään READ_SECONDS, jotta sen arvo saadaan pienemmäksi kuin READ_SECONDS-makroon määritetty luku, mutta mahdollinen ylimääräinen kertymä kahdeksan sekunnin ajastimesta jää talteen. Silmukan lopussa laite laitetaan aina lepotilaan kutsumalla powerdown-funktiota. Arduino kutsuu loop-funktiota toistuvasti setup-funktion suorittamisen jälkeen, jonka takia funktiossa ei tarvitse käyttää erillistä silmukkaa.

Arduinolle ladatussa ohjelmassa (esimerkkikoodit 2 – 7) hyödynnetään Watchdog-ajastinta laitteen virransäästämistä ja kosteusmittarin lukemisen ajastamista varten. Watchdog-ajastinta käytettäessä laite voidaan laittaa vähävirtaisimpaan lepotilaan. Unossa oleva ATmega328P tukee kuutta erilaista lepotilaa, joista ”Power-down” on kaikista vähävirtaisin. Power-down-tilasta prosessorin voi herättää mm. Watchdog-ajastimella tai ulkoisella keskeytyksellä [24]. Ohjelmaan määritetyllä READ_SECONDS-makrolla määritetään suurin piirteinen lukutiheys sekunneissa. Lukutiheys ei ole tarkka, koska ohjelmassa käytetään aina kahdeksan sekunnin mittaista virransäästöjaksoa eikä ohjelmassa yritetä kompensoida muista syistä, kuten ajastimen epätarkkuudesta, johtuvia siirtymiä.

Prototyyppiä testattiin pikaisesti 200 mAh 9 V:n akulla. Ensimmäisessä testissä akun jännite laski alle toimintarajan noin 10 min käytön jälkeen. Ensimmäisessä testiversiossa ei käytetty mitään virransäästöominaisuuksia. Seuraavaksi testattiin esimerkkikoodeissa 2 – 7 esiteltyä ohjelmaa. Pelkällä power-down-tilan käyttöönotolla ja turhien prosessorin laitteiden käytöstä poistamisella laitteen käyttöaika kasvoi noin kolmeen ja puoleen tuntiin. Laitteen virrankulutusta voisi laskea helposti lisää katkaisemalla virran DHT22-anturilta ja Bluetooth-moduulilta, kun ne eivät ole käytössä. Virran katkaiseminen vaatisi pieniä muutoksia piiriin ja ohjelmaan.

5.2.2 Kosteuden valvonta

Humidorin kosteus- ja lämpötilamittarin valvonta toteutettiin hyödyntämällä kahta ohjelmaa. Ohjelmia testattiin vain Ubuntu-Linuxilla. Tietojen lukuohjelma lukee sille argumenttina annetulta sarjalaitteelta JSON-viestejä. Viesteihin lukuohjelma lisää aikaleiman ja

laitteen nimen ennen niiden lähettämistä palvelinohjelmalle. Tietojen lähettämiseen käytetään ZMQ-kirjastoa. Palvelinohjelma vastaanottaa lukuohjelmien viestit ja tallentaa ne mongo-tietokantaan. Palvelinohjelma käynnistää toiseen prosessiin HTTP-API:n valvontaa varten. API toteutettiin käyttäen flask-kirjastoa. API tarjoaa kaksi rajapintaa, joista molemmat palauttavat vastaukset JSON-muodossa. Palvelimen juuri (/) palauttaa listan tietokannasta löytyvistä laitteista ja /laitteen_nimi palauttaa laitteelle JSON-tiedoston, joka sisältää laitteen nimen, viimeisimmät tiedot kosteudesta ja lämpötilasta, mitkä laitteella on mitattu sekä aikaleiman viimeisimmästä mittauksesta.

```
#!/usr/bin/env python3
import json, time, math, argparse, requests

def get_data(host, port, sensor):
    r = requests.get("http://{}:{}/{}".format(host,port,sensor))
    if r.status_code != 200:
        raise Exception("Failed to load data")
    rd = r.content.decode("utf-8")
    d = json.loads(rd)
    return d

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--host", default="127.0.0.1")
    parser.add_argument("--port", default=7777)
    parser.add_argument("--warn-humidity", "-w", type=float,
                        default=3,
                        help="Warning rh%% offset from 70%%")
    parser.add_argument("--crit-humidity", "-c", type=float,
                        default=5,
                        help="Critical rh%% offset from 70%%")
    parser.add_argument("--timelimit", "-t", type=int,
                        default=120)
    parser.add_argument("sensor")
    try:
        args = parser.parse_args()
    except (argparse.ArgumentError, SystemExit):
        exit(3)

    try:
        data = get_data(args.host, args.port, args.sensor)
        now = int(time.time())
        if now - data["ts"] > args.timelimit:
            print("Unknown: Data on server too old! ({}s since
last update)".format(now - data["ts"]))
            exit(3)
        rh_offset = math.fabs(70 - data["rh"])
        if rh_offset > args.crit_humidity:
            print("Critical: {sensor} humidity @
{rh}!".format(**data))
            exit(2)
        elif rh_offset > args.warn_humidity:
            print("Warning: {sensor} humidity @
{rh}!".format(**data))
            exit(1)
        print("OK: {sensor} humidity @ {rh}!".format(**data))
```

```
        exit(0)
    except Exception as e:
        print("UNKNOWN: Unable to fetch data or data in wrong
format!")
    exit(3)
```

Esimerkkikoodi 8. Kosteuden valvontaskripti

Valvontajärjestelmää varten tehdyllä valvontaskriptille (esimerkkikoodi 8) annetaan vähintään laitteen nimi argumenttina. Muille parametreille on annettu oletusarvot, joten niitä ei ole pakko antaa. Skripti pyytää palvelinohjelman API:lta laitteen viimeisimmät tiedot /laitteen_nimi- polusta. Vastauksena saadun JSON-tiedoston kosteutta ja aikaleimaa verrataan valvontaskriptille argumentteina annettuihin raja-arvoihin tai oletusarvoihin. Skriptille kosteuden raja-arvot annetaan poikkeamana seitsemänkymmenen prosentin suhteellisesta kosteudesta. Oletusarvoisesti skripti palauttaa varoituksen, kun kosteus poikkeaa yli kolmella prosenttiyksiköllä seitsemästäkymmenestä ja kriittinen palautetaan yli viiden prosenttiyksikön poikkeamista. Tuntematon palautetaan, kun skripti ei saa tietoja halutulle anturille, tai tietojen kyselyhetken aikaleiman ja JSON:in aikaleiman erotus ylittää argumenttina annetun, tai oletusarvoisen, rajan. Oletusarvo aikaleiman tuoreudelle on kaksi minuuttia.

Teknologiat valittiin mukaillen yrityksen käyttämiä teknologioita, jotta opiskelisin niiden käyttöä henkilökohtaisesti hieman erilaisessa kontekstissa. Projektin arkkitehtuurikin mukailee yrityksen anturidatan välittämiseen ja käsittelyyn hyödynnettyä järjestelmää siinä mielessä, että molemmissa keskeisessä roolissa on ZMQ-viestijono (luku 4.1). Monet asiat olisi voitu tehdä yksinkertaisemminkin, jos olisi haluttu toteuttaa vain palvelu kosteuden valvontaa varten. Muun muassa tiedot voitaisiin tallentaa tietokantaan jo lukuohjelmassa.

Valvontajärjestelmän tarkastus konfiguroitiin prototyyppiä testattaessa ajettavaksi kerran minuutissa. Tarkastus konfiguroitiin aiheuttamaan hälytyksiä vain viiden tai useamman epäonnistumisen jälkeen. Useampi epäonnistuminen asetettiin vaatimukseksi, jotteivat lyhytaikaiset muutokset aiheuta hälytyksiä. Kosteus voi esimerkiksi laskea hetkellisesti, kun humidori avataan. Jos anturi otettaisiin oikeasti käyttöön kosteuden valvontaa varten, mittausaikaväliä ja tarkastusten suoritusaikaväliä olisi hyvä pidentää. Normaalisti kosteus muuttuu hitaasti, joten tiiviillä mittaamisella ei saavutettaisi oleellista hyötyä.

Valvontajärjestelmän konfiguroinnin kannalta tietokoneeseen suoraan kytkettävät anturit ovat hieman hankalia. Koska anturin toimintaa voidaan tarkkailla vain tietokoneelta, johon se on kytketty. Valvonta agentittomasti on kuitenkin suhteellisen helposti konfiguroitavissa, mutta automatisointi ja skaalautuvuus muuttuu vaikeammaksi, elleivät järjestelmät ole keskenään samanlaisia. Kehittämällä keskitetty antureita hyödyntävä järjestelmä, jossa on tuki valvontaa varten, tekee valvontajärjestelmän tarkastuksien konfiguroinnista helpompaa. Keskitetyllä ratkaisulla valvontajärjestelmän ei tarvitse olla tietoinen, mihin tietokoneeseen mikäkin antureista on kytketty. Keskitetyllä ratkaisulla valvontavastuukin voidaan jakaa useammalle agentille. Kaikista parasta valvonnan kannalta olisikin käyttää mieluummin verkkoon kytkettävissä olevia antureita.

6 Tulokset

Yrityksen käyttöön valitusta valvontajärjestelmästä oli hyötyä jo osittaisen käyttöönoton jälkeen. Monitoroinnin avulla löydettiin mm. muistivuoto eräästä tuotantokäytössä olevasta ohjelmasta ja saatiin selville vuodon nopeus hyödyntäen hälytysten aikaleimoja. Löydön pohjalta muistivuodon syytä pystyttiin alkaa selvittää. Väliaikaisena ratkaisuna ohjelma ajastettiin käynnistymään uudelleen päivittäin, jotta muistivuodosta ei olisi haittaa muille palvelimella ajettaville ohjelmille.

Valvontajärjestelmää pystyttiin käyttämään apuvälineenä vikatiedotteiden laatimisessa. Opinnäytetyön projektiosan tekemisen aikana järjestelmän lähettämiä viestijä hyödynnettiin ainakin kahdesti eri asiakkaiden vikatiedotteissa. Molemmissa tapauksissa asiakkaiden verkoista oli irrotettu tärkeitä verkkolaitteita. Hälytysten ansiosta asiakkaisiin pystyttiin ottamaan yhteyttä heti hälytysten tultua. Nopean toiminnan ansiosta asiakkaat pystyivät selvittämään irrotusten syyt ja saamaan laitteet takaisin verkkoon ennen kuin vahinkoa ehti syntymään.

Valvontajärjestelmän kehittäminen jatkuu vielä. Joitakin tuotannon ohjelmia ei pystytä vielä monitoroimaan tehokkaasti, koska ohjelmien tilaa ei voida tarkistaa ohjelman ulkopuolelta. Jatkokehityksen tavoitteina on kehittää osaan monitoroitavista ohjelmista toimintotilan kyselemistä varten.

Arduino-prototyypin kehittämisessä saavutettiin halutut tavoitteet. Prototyyppi on mahdollista siirtää muille Arduino-yhteensopiville alustoille riittävän helposti. Tulevaisuudessa mittari voidaan siirtää fyysiseltä koolta pienemälle ja vähävirtaisemmalle alustalle.

Bluetooth-moduuli vaihdetaan joko toiseen vähävirtaisempaan vaihtoehtoon tai Bluetoothin tilalle valitaan jokin toinen langaton tiedonsiirtotapa. Mittari tulee tulevaisuudessa toimimaan paristolla. Tavoitteena on saada se toimimaan vähintään vuosi ilman pariston vaihtoa. Lopuksi mittarille on tarkoitus tulostaa kotelo 3D-printterillä. Mittarin tulevaksi alustaksi harkittiin Arduino MKRZeroa ja JeeNode Zeroa. Molemmat alustat ovat vähävirtaisia. Lopullisia päätöksiä mittarin kehittämiselle ei tehty.

Tietokoneeseen liitettävien antureiden monitoroiminen tavallisten verkonvalvontajärjestelmien avulla on mahdollista. Käytännössä yksittäisten tietokoneeseen kytkettyjen laitteiden valvominen valvontajärjestelmällä agentittomasti ei kuitenkaan vaikuttanut erityisen järkevältä siihen nähtävän vaivan takia. Mikäli yritys teetättää antureita omaan käyttöönsä, antureiden olisi hyvä olla verkkoon kytkettävissä, jos niitä halutaan monitoroida suoraan valvontajärjestelmällä. Tietokoneeseen suoraan kytkettyjä laitteita on parempi valvoa niitä hyödyntävien järjestelmien kautta.

7 Yhteenveto

Opinnäytetyö jakautui kahteen osaan. Ensimmäisessä osassa valittiin yrityksen käyttöön verkonvalvontajärjestelmä, otettiin se käyttöön ja kehitettiin sitä varten valvontaskriptejä.

Valvontaskriptien kirjoittamista varten tutustuttiin Nagioksen valvontaskriptien API:iin. API:iin tutustuttiin, koska yrityksen käyttöön valittu valvontajärjestelmä (Sensu) käyttää Nagioksen valvontaskriptien kanssa yhteensopivaa API:a.

Toisessa osassa kehitettiin Arduino-pohjainen prototyyppi humidorin kosteuden valvontaa varten, ja testattiin sen valvontaa yrityksen järjestelmällä. Prototyypin kytkemiseksi valvontajärjestelmää kehitettiin muutama pieni ohjelma ja valvontaskripti kosteuden tarkastamista varten.

Lähteet

1 Slawek Ligus. 2013. Effective Monitoring and Alerting. O'Reilly Media, Inc.

2 Why do my computer systems need monitoring? Verkkoaineisto. <https://www.swicktech.com/SWICKtech/Resources/Blog/Why-do-my-computer-systems-need-monitoring.htm> Luettu 4.4.2018.

3 What is Agentless Monitoring? Verkkoaineisto. <https://www.eginnovations.com/product/agentless-monitoring>. Luettu 4.4.2018.

4 Passive checks. Verkkoaineisto. < https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/passivechecks.html>.

5 Handlers. Verkkoaineisto. <https://docs.sensu.io/sensu-core/1.2/reference/handlers/>. Luettu 4.4.2018.

6 Filters. Verkkoaineisto. <https://docs.sensu.io/sensu-core/1.0/reference/filters/>. Luettu 4.4.2018.

7 History of Nagios. Verkkoaineisto. <https://www.nagios.org/about/history/>. Luettu 4.4.2018.

8 Nagios Exchange. Verkkoaineisto. <https://exchange.nagios.org/directory/Plugins>. Luettu 4.4.2018.

9 Evaluation of NoSQL databases for DIRAC monitoring and beyond. Verkkoaineeisto. <https://cds.cern.ch/record/2011172/files/LHCb-TALK-2015-060.pdf>. Luettu 10.10.2016.

10 Sensu Github, ensimmäinen commit. Verkkoaineisto. <https://github.com/sensu/sensu/commit/7bb7f2887f8677cd51e74e23b6ad96be5f47e8b3>. Luettu 4.4.2018.

11 Clustering Guide. Verkkoaineisto. <https://www.rabbitmq.com/clustering.html>. Luettu 4.4.2018.

12 Redis cluster tutorial. Verkkoaineisto. <https://redis.io/topics/cluster-tutorial>. Luettu 4.4.2018.

13 Sensu Frequently Asked Questions. Verkkoaineisto. <https://docs.sensu.io/sensu-core/1.2/faq/#sensu-frequently-asked-questions >. Luettu 4.4.2018.

14 Nagios Plugin API. Verkkoaineisto. <https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/pluginapi.html>. Luettu 8.12.2017.

15 Exit Codes With Special Meanings. Verkkoaineisto. <http://tldp.org/LDP/abs/html/exitcodes.html>. Luettu 8.12.2017.

16 Out Of Memory Management. Verkkoaineisto. <https://www.kernel.org/doc/gorman/html/understand/understand016.html>. Luettu 4.4.2018.

17 Nagios Plugins Development Guidelines. Verkkoaineisto. <https://nagios-plugins.org/doc/guidelines.html>. Luettu 12.12.2017.

18 User parameters. Verkkoaineisto. <https://www.zabbix.com/documentation/3.4/manual/config/items/userparameters>. Luettu 8.12.2017.

19 Mike Julian. 2018. Practical Monitoring, Effective Strategies for the Real World. O'Reilly Media, Inc.

20 Richard Carleton Hacker. 2015.The Ultimate Cigar Book: 4th Edition. Skyhorse Publishing. (kappale 5, s. 1-4)

21 Huoneen optimaalinen kosteus. Verkkoaineisto. <http://www.rytmirakennus.fi/2012/01/huoneen-optimaalinen-kosteus/>. Luettu 2017.05.23.

22 DHT22 temperature-humidity sensor + extras. Verkkoaineisto. <https://www.adafruit.com/product/385>. Luettu 4.4.2018.

23 Bluetooth Module HC-05. Verkkoaineisto. <https://wiki.eprolabs.com/index.php?title=Bluetooth_Module_HC-05>. Luettu 4.4.2018.

24 ATmega328/P Datasheet. 2016. Atmel Corporation.